Routing in Multicomputer Networks: A Classification and Comparison

by

Farooq Ashraf

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

June, 1996

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313/761-4700 800/521-0600

ROUTING IN MULTICOMPUTER NETWORKS: A CLASSIFICATION AND COMPARISON

BY

FAROOQ ASHRAF

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
In

COMPUTER SCIENCE

JUNE, 1996

ROUTING IN MULTICOMPUTER NETWORKS: A CLASSIFICATION AND COMPARISON

MS Thesis

Farooq Ashraf

Information & Computer Sciences

June 1996

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by **FAROOQ ASHRAF** under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate Studies, in partial fulfillment of the requirements for the degree of **MASTER OF SCIENCE** in **COMPUTER SCIENCE.**

THESIS COMMITTEE

Dr. Khalid Al-Tawil (Chairman)

Dr. Mostafa Abd El-Barr (Co-Chairman)

Dr. Muhammed Al-Mulhem (Member)

Dr. Muslim Bozyigit (Member)

Department Chairman

Dean, College of Graduate Studies

Date: 30.6.96

COLLEGE OF GRADUATE STUDIES

Dedicated to

My Parents

Acknowledgments

All Praise is due to Allah Almighty, the Lord of the Worlds, and peace and blessings be upon His Last Prophet Muhammad, his family, companions and followers till the Day of Judgement.

I have been fortunate to have the opportunity to work under the supervision of Dr. Khalid Al-Tawil and Dr. Mostafa Abd-El-Barr. Their valuable suggestions, continuous encouragement and extraordinary support throughout all stages of this research are highly appreciated. I would like also to place on record my appreciation for the cooperation and guidance extended by my committee members, Dr. Muhammed Al-Mulhem and Dr. Muslim Bozyigit.

I would like to thank all my friends, both from within and outside the department, for making my stay at KFUPM a pleasant and memorable one.

Contents

List of Figures

List of Tables

ABSTRACT

Name: FAROOQ ASHRAF

Title: ROUTING IN MULTICOMPUTER NETWORKS: A CLASSIFICATION AND COMPARISON

Degree: MASTER OF SCIENCE

Major Field: INFORMATION & COMPUTER SCIENCE

Date of Degree: JUNE 1996

Multicomputer systems are distributed-memory MIMD systems where the processing elements communicate via the interconnection links. The method used to choose the path for a message over the multicomputer links is referred to as routing. An extremely wide number of routing algorithms have been proposed and implemented in hardware and software in the last two decades. However, these have not been collectively studied todate. There have been surveys that are limited in scope. Some concentrate on a particular topology while others on a specific switching technique. We felt a genuine need for collecting together the various routing techniques for multicomputer networks proposed in the literature, classifying them and comparing them. In this study we propose a classification and characterization model for routing schemes in multicomputer networks. We also study a number of routing techniques for one-to-one routing, fault-tolerant one-to-one routing, and broadcast and multicast routing. Moreover, we present a comparison between the various classes of routing techniques. The comparison has been based on the proposed classification, the topology and other features such as the port model, complexity, and the size of the routing path. The study can be helpful to a designer of multicomputer networks by providing an overview of the available techniques, their classification and relative advantages and disadvantages. Thus based on this study, better course of action may be adopted during the design phase.

Master of Science Degree
King Fahd University of Petroleum and Minerals, Dhahran.
June 1996

خلاصة الرسالة

اسم الطالب الكامل : فاروق أشرف

عنوان الدراسة : طرق التسيير في شبكات الحاسبات المتعددة : تصنيف ومقارنة

التخصص : علوم الحاسب والمعلومات

تاريخ الشهادة : يونيو (حزيران) ١٩٩٦ م - صفر ١٤١٧ هـ

تتصل المعالجات في شبكات الحاسبات المتعددة ببعضها البعض عن طريق روابط خاصة بين الوحدات. و تسمى عملية اختيار طريق الاتصال داخل هذه الشبكات "*التسيير*". و خلال العقدين الماضيين تم اقتراح العديد من طرق التسيير في شبكات الحاسبات. و من المعلوم أن هذه الطرق لم تدرس متكاملة إلى الآن، لكن بعض الدراسات قد اجريت في مجال محدود، فبعضها يركز على مواقع الالتقاء في الشبكات بينما يركز بعضها الآخر على طرق توصيل البيانات. لذلك احسسنا باهمية جمع هذه الطرق المختلفة للتسيير في انظمة الحاسبات المتعددة وتصنيفها ومقارنة هذه الطرق بعضها ببعض. وفي هذا البحث نقترح طريقة خاصة لتصنيف الطرق المعتمدة في التسيير في أنظمة الحاسبات المتعددة. وبالاضافة إلى ذلك، تم دراسة كل من طرق التسيير الأحادية وطرق التسيير الأحادية في حالة وجود خلل او عطل جزئي في الشبكة وكذلك تم دراسة طرق التسيير من وحدة إلى جميع الوحدات او بعض الوحدات في الحالات العادية، او في حالة وجود خلل في الشبكة. كما اننا نقدم في هذه الدراسة مقارنة بين طرق التسيير المختلفة. و لقد اعتمدنا في تصنيف هذه الطرق على التصنيفات التي تعتمد على درجة التعقيد وحجم الطرق المستخدمة في التسيير ونوع الاطراف المستخدمة في هذه الشبكات، بالاضافة إلى الطرق الاخرى. وتعتبر هذه الدراسة مفيدة لمصممي شبكات الحاسبات المتعددة بإعطائهم نظرة عامة عن الطرق الموجودة و تصنيفها وميزات كل منها بالاضافة إلى العيوب الموجودة في بعض هذه الطرق. إننا لنأمل من هذه الدراسة المستفيضة أن يتم التوصل إلى طرق جديدة وفعّالة في تصميم شبكات الحاسبات المتعددة .

درجة الماجستير في العلوم

جامعة الملك فهد للبترول والمعادن
الظهران، المملكة العربية السعودية

يونيو (حزيران) ١٩٩٦م - صفر ١٤١٧هـ

Chapter 1

Introduction

The growing demand for high processing power in various scientific and engineering applications has made multiprocessing architectures increasingly popular. One of the design methodologies used for parallel machines has led to the development of distributed-memory message-passing concurrent computers, commonly known as ***multicomputers***. They consist of many processing nodes that interact by sending messages (containing both data and synchronization information) over communication links. The nodes are connected together by interconnection networks in various topologies such as meshes, hypercubes, etc.

All multicomputers depend on the underlying interconnection network to communicate data between individual processing elements. An interconnection network is characterized by its topology, switching, flow control and routing. The ***topology*** defines how the nodes are interconnected by links and is usually modeled as a graph. ***Switching technique*** is the actual information transmission mechanism that facilitates moving data through the network. ***Flow control*** of a network determines

how resources such as buffers and channel bandwidth are allocated and how packet collisions are resolved. A resource collision occurs when a packet is unable to proceed because any resource it needs is held by another packet. Whether the packet is buffered, blocked in place, dropped or misrouted depends on the flow control policy. A good flow control policy should avoid channel congestion while reducing the latency. ***Routing*** specifies the path of a packet through the network. For maximum system performance, a routing algorithm should have high throughput and low latency, avoid deadlock, livelock and starvation, and be able to work well under various traffic patterns.

Providing efficient communication between the nodes of a multicomputer network is one of the most important problems in parallel computing and it depends upon the underlying routing scheme. For this reason it is essential to know which routing techniques are suitable and practical. Although an extremely wide number of routing algorithms have been proposed and implemented in hardware and software, it is difficult for a designer of multicomputers to choose an appropriate routing algorithm given a particular architectural configuration. For this purpose there is a need for collecting together the various techniques proposed in the literature, classifying them and comparing them. Such an extensive study has not been done previously. In fact there have been only two surveys [46, 107] published recently. The first of these deals only with routing techniques for the hypercube topology, whereas the second one considers only wormhole switching based routing techniques. However, these two papers do provide a good introduction to the area of multicomputer routing.

1.1 Motivation

In view of the lack of a comprehensive study of multicomputer routing techniques and the following shortcomings of the available studies, we have presented a detailed discussion of uptodate techniques along with a classification and comparison scheme of the same.

1. Most of the available studies focus on a particular type of architecture studying routing techniques, for example, only for the hypercube topology as pointed out above.

2. Some only study techniques designed for a particular switching mechanism also mentioned above.

3. Most have overlooked the fault-tolerant routing techniques.

4. Not all classes of the classification hierarchy have been studied.

5. The theoretical background necessary for studying routing in multicomputer networks has not been presented in a consolidated and lucid manner.

6. A collective study of routing techniques for multicasting and broadcasting schemes, that have become widely used during the last decade or so, has not been done.

1.2 Objective

Based on the above-mentioned observations and feeling the need for an exhaustive study that could help the designers and implementors of multicomputer systems, a fresh study of routing strategies is proposed with the following objectives:

1. To present a classification model for routing techniques.
2. To study the techniques on different architectures, e.g., hypercubes, meshes, and k-ary n-cubes, and with various switching mechanisms.
3. To study the fault-tolerant routing algorithms on various architectures and with different switching techniques.
4. To survey the literature extensively and classify the published work according to the proposed classification and present a comparison.
5. To provide the conceptual basis necessary for conducting further study in the area of routing in multicomputer networks.
6. To study and compare various routing techniques for multicast and broadcast communication in detail.

1.3 Organization of Thesis

This thesis is organized as follows. In chapter 2 an introduction to multicomputer networks and their various characteristics are presented. Chapter 3 presents the

proposed classification of routing techniques. Various switching techniques are introduced in Chapter 4. The concept of virtual channels is discussed in Chapter 4. Chapter 5 presents routing techniques for one-to-one communication. Fault-tolerant routing techniques are surveyed in Chapter 6. Routing techniques for broadcast and multicast communication are studied in Chapter 7. Chapter 8 concludes the thesis with suggestions for extending this research.

Chapter 2

Multicomputer Networks

Distributed-memory loosely coupled MIMD systems are called ***multicomputer systems*** [32]. The processing elements in these multicomputers must communicate by explicitly passing messages, so they have also been defined as message-passing highly/massively concurrent/parallel computers. These systems are organized as ensembles of small programmable processing elements (which maybe powerful computers with full computing capabilities), called ***nodes***. These nodes are connected by a message-passing network, which together with its processing elements is called the ***multicomputer network***. The term "***multicomputer***" comes from "***multi***ple-***computer***" which actually depicts the structure of such systems. A general representation of multicomputer networks is given in Figure 2.1.

Multicomputers are both logically and physically distributed-memory systems. The processor in each node is tightly coupled to a memory that is physically separate and logically private from the memories of all the other nodes. A global memory address does not exist, rather, each node has its own private memory address space.

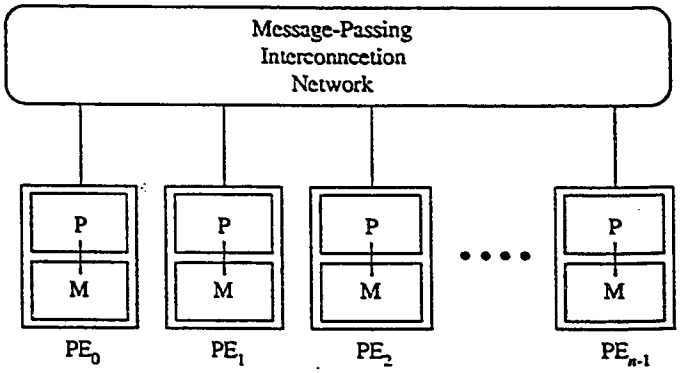


Figure 2.1: A general representation of an n-node multicomputer.

The fixed numbering of the nodes, $0, 1, \ldots, n-1$, together with the unique numbering of the processes within each node establish globally unique identifiers for processes; hence, a global name space.

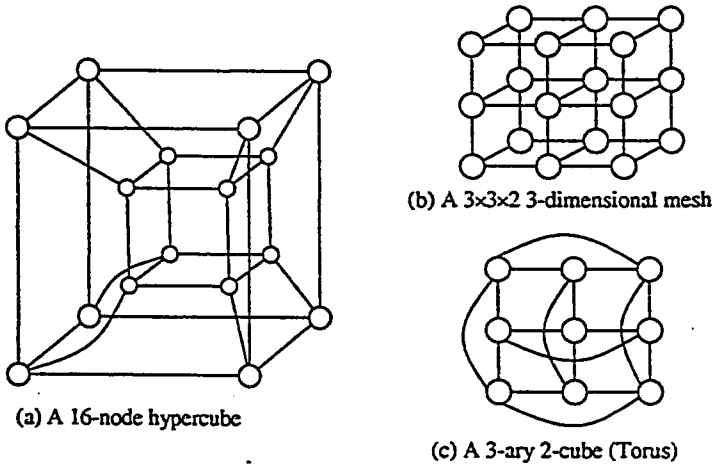


Figure 2.2: Direct network topologies.

Interprocess/interprocessor communication in a multicomputer occurs by routing messages through a network such as a binary n-cube or mesh, (see Figure 2.2). Regular networks simplify the routing. Their direct and bidirectional, as opposed to indirect, structure allows process placement to exploit the locality of communica-

tions. Some of these networks are also ***extensible*** or ***scalable***, to allow for systems with different number of nodes. The individual channels operate at rates comparable to the memory bandwidth of a node; the nodes cannot generate or absorb messages at a higher rate.

Although no single or perfect definition of ***grain size*** exists for multicomputers, the term loosely describes node size or complexity. Multicomputers whose nodes contain megabytes of memory are referred to as ***medium-grain*** machines, and those whose nodes contain tens of kilobytes of memory as ***fine-grain*** machines.

If all of the nodes of a multicomputer are same, the machine is called ***homogeneous***. If mixed types of nodes are present then it is referred to as ***heterogeneous***. Because multicomputer processes are distributed across the nodes, their operating system support is best described as ***distributed multiprogramming*** [3].

Multicomputer networks are either indirect or direct. In the present work we restrict our attention to the latter only. However, for the sake of completeness, a brief introduction to the former is provided below.

2.1 Indirect Networks

In indirect networks, the switching nodes are distinct from the processing nodes. They are called ***indirect*** because messages between processing nodes are routed indirectly through the switching nodes. Figure 2.3 shows two examples of this class of networks.

The nodes on the left and right sides of the network are the processing nodes. In some machines the network is folded back on itself so that the nodes on the left

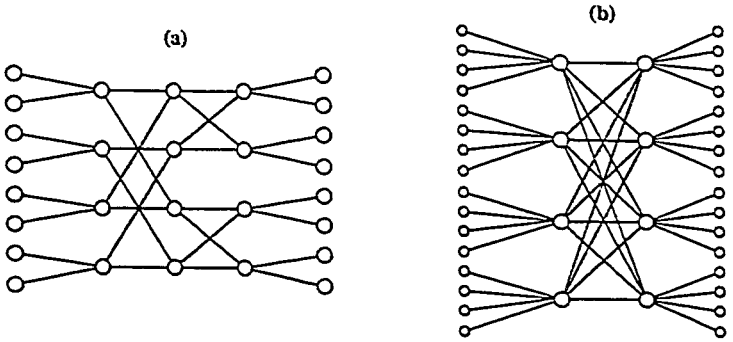


Figure 2.3: Indirect Networks (a) a 2-ary 3-fly (b) a 4-ary 2-fly.

and right are the same. In other machines, processor nodes are placed on one side of the network, and memory nodes are placed on the other side. The nodes in the center of the network are the switching nodes.

The networks shown in Figure 2.3 are radix k, dimension n butterflys. They are usually referred to as k-ary n-flys. A k-ary n-fly has $N = k^n$ processing nodes. Processing node i is connected to an input node $\lfloor i/k \rfloor$ in the first stage of the switch and an output of the corresponding node in the last stage of the switch. Output port j of node i in the s^{th} stage of the switch is connected to an input port of node $(i + jk^{n-s-1}) \mod N/k$. A 2-ary 3-fly (8 nodes) is shown in Figure 2.3(a). A 16-node 4-ary 2-fly is shown in Figure 2.3(b) [26]. An indirect network is characterized by the same features as those of the direct networks. Anyone further interested in the area of indirect networks may refer to [1, 6, 60, 85, 124, 125].

2.2 Direct Networks

In a direct network, all switching is performed in the processing nodes. The prototypical direct network is k-ary n-cube, which includes the various popular topologies

such as hypercubes, tori, etc.

There are many ways to interconnect nodes in a direct network. ***topology*** refers to the interconnection graph of the network $I = G(N, C)$. The vertices of this graph are the nodes of the network, N, and the edges are the physical channels that connect the nodes, $C \subset N \times N$. Most of the popular direct network topologies fall in the general category of either n-dimensional meshes or k-ary n-cubes because their regular topologies simplify routing.

Formally, an n-dimensional mesh has $k_0 \times k_1 \times \ldots \times k_{n-2} \times k_{n-1}$ nodes, k_i nodes along each dimension i, where $k_i \geq 2$. Each node x is identified by n coordinates, $\sigma_{n-1}(x), \sigma_{n-2}(x), \ldots, \sigma_1(x), \sigma_0(x)$, where $0 \leq \sigma_i(x) \leq k_i - 1$ for $0 \leq i \leq n-1$. Two nodes x and y are neighbors if and only if $\sigma_i(x) = \sigma_i(y)$ for all i, $0 \leq i \leq n-1$, except one, j, where $\sigma_j(y) = \sigma_j(x) \pm 1$. Thus, nodes have from n to $2n$ neighbors, depending on their location in the mesh.

In a k-ary n-cube, all nodes have the same number of neighbors. The definition of a k-ary n-cube differs from that of an n-dimensional mesh in that all of the k_i's are equal to k and two nodes x and y are neighbors if and only if $\sigma_i(x) = \sigma_i(y)$ for all i, $0 \leq i \leq n-1$, except one, j, where $\sigma_j(y) = (\sigma_j(x) \pm 1) mod k$. The use of modular arithmetic in the definition results in ***wraparound*** channels in the k-ary n-cube, which are not present in the n-dimensional mesh. A k-ary n-cube contains k^n nodes. If $k = 2$, then every node has n neighbors, one in each dimension. If $k > 2$, then every node has $2n$ neighbors, two in each dimension. A k-ary n-cube is a radix k cube with n dimensions. The ***radix*** implies that there are k nodes in each dimension. In general, a k-ary n-cube is constructed from k k-ary $(n-1)$-cubes

by connecting corresponding elements of the two into k-rings. Every node has an address that is an n digit radix k number. Nodes are connected to all nodes with an address that differs in only one digit [107].

Several special cases of the n-dimensional meshes and k-ary n-cubes have been proposed or implemented as direct network topologies. When $n = 1$, the k-ary n-cube collapses to a ring with k nodes. The hypercube is a symmetric network and a special case of the k-ary n-cube. Figure 2.2(a) depicts a binary 4-cube, or 16-node hypercube. When $n = 2$, the topology is a 2D torus with k^2 nodes. Figure 2.2(c) illustrates a 3-ary 2D torus. Figure 2.2(b) shows a $3 \times 3 \times 2$ 3D mesh.

Hypercubes, low-dimensional meshes, and tori can be compared in terms of their bisection width η, that is, the minimum number of channels that must be removed to partition the network into two equal subnetworks. Both the hypercube and the torus are symmetric networks in that there exists a homomorphism that maps any node of the graph representing the network graph onto any other node. Mesh networks, on the other hand, are asymmetric because the wraparound channels are absent. Assuming uniform traffic between nodes, channels near the center of the mesh are likely to experience higher traffic density than channels on the periphery. In addition, the network diameter is doubled. On the other hand, it is easier to provide deadlock-free routing in mesh networks than in torus networks. Furthermore, in a 2D layout, long wraparound wires connecting boundary nodes are eliminated to further reduce the wire complexity and reduce the bisection width by half.

Further detail on topological properties of multicomputer networks could be found in [34, 76, 107, 115, 119, 127].

Characteristics of Direct Networks

A direct network is characterized by four factors: topology, routing, flow control, and switching.

Topology

A multicomputer topology is evaluated in terms of the following five parameters [26].

- **Bisection Width :** The ***channel bisection***, B, is the minimum number of channels that, when cut, separate the network into two equal parts. The ***wire bisection***, B_W, is the number of wires crossing this cut of the network. $B_W = BW$ where W is the width of a channel in bits. ***Bisection width*** measures how much wiring density is required by a network. For design purposes, wire bisection is fixed, and the bisection width constraints how wide each channel can be: $W = B_W/B$.

- **Degree :** The ***degree*** of a node, d, is the number of channels incident on the node. The number of channels into the node is the ***in degree*** of the node, d_{in}. The number of channels out of a node is the ***out degree***, d_{out}^{l}. The total degree is the sum, $d = d_{\text{in}} + d_{\text{out}}$. The degree affects the cost of a node. The number of pins on each node is Wd, and the complexity of the router's control logic is usually related to d.

- **Diameter :** The ***diameter*** of a network is the longest shortest path between any two nodes.

$$D = \max_{i,j \in N} \left(\min_{p \in P_{i,j}} \text{length}(p) \right),$$

where $P_{i,j}$ is the set of paths from i to j. Diameter has traditionally been used as the primary figure of merit for networks. The shorter the diameter the better the performance. Hence the popularity of networks such as binary n-cubes with low diameters.

To find the diameter, consider all pairs of nodes $i, j \in N$ and find the shortest path from i to j. Then take the maximum of this path length over all pairs of points.

- **Wire Length :** The length of the wires in the network determines the speed at which the network can operate and the amount of power dissipated driving these wires.

- **Symmetry :** A network is symmetric if it is isomorphic to itself with any node labeled as the origin. In a symmetric network the network ***looks*** the same from every node. Rings and tori are symmetric, while linear arrays and meshes are not. Symmetric networks simplify many resource management problems.

Routing

Routing is the method used for a message to choose a path over the network channels. Routing and its various classifications are further described in Chapter 3.

Flow Control

Flow control refers to the method used to regulate traffic in a network. It determines when a message or part of a message can advance. Flow control is the

resource management policy that is used to allocate communication resources, i.e., wires, and buffers, to information units, messages, packets, and flits. A ***policy*** is required to determine which circuits must be throttled, and when, in order to make the network capable of "throttling" the traffic on the various channels, to prevent buffer overflow and to handle situations when a processor is sent more messages than it can immediately receive. This policy is provided by the flow control mechanism. A good flow control policy should avoid channel congestion while reducing the network latency.

- ***Message :*** It is the logical unit of communication. Two objects communicate by sending messages. This is the only unit seen by clients of the network service.

- ***Packet :*** A message is divided into one or more packets. A packet is the smallest unit that contains routing information – e.g., the destination address. If there is more than one packet in a message, each packet also contains a sequence number to permit reassembly. Long messages must be broken into many packets to avoid degrading network performance.

- ***Flit :*** A packet can be further divided into flow control digits or ***flits***, the smallest unit on which flow control is performed. That is, communication resources are allocated on a flit basis. In general, a flit contains no routing information. Only the ***leading*** flit of a packet knows where it is going.

We use the general term ***message*** to mean packet, message (in the original sense of the word) or flit, whichever is appropriate in a context. Figure 2.4 illustrates the

resources that flow control has to allocate. Actual communication is performed by the physical channel. Buffers are used to store flits before and after they use the channel. Some channel state is required to remember the route for nonleading flits traversing the channel. A source buffer, a destination buffer, and some channel state make up a ***virtual channel.*** It is a channel from the point of view of resource allocation, but may share a physical communication channel with other virtual channels. When the leading or head flit of a packet arrives at a channel, it requests a virtual channel. Once the channel is allocated, it will remember the next step of the route (i.e., which outgoing channel to use) until the tail flit of the packet is encountered. Starting with the head flit, each flit in turn must now compete for buffer and channel resources. For a flit to advance, three conditions must hold: (1) It must be resident in a source buffer; (2) it must be allocated an empty destination buffer; and (3) it must be allocated use of the physical channel [26].

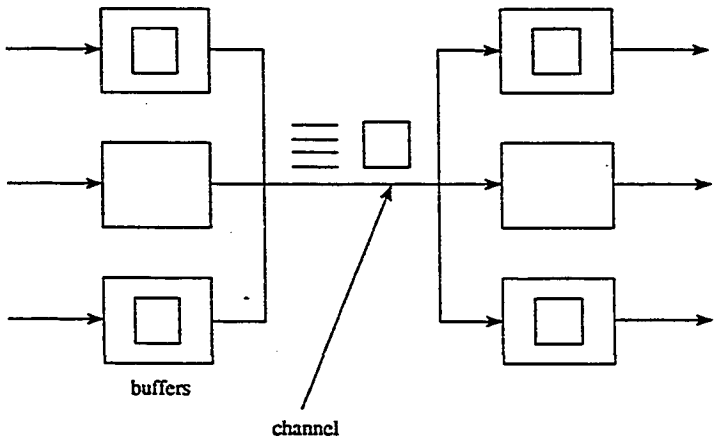


Figure 2.4: A channel consisting of the physical channel, source buffers, destination buffers and some channel state.

Flow control methods are distinguished by how they resolve collisions between packets. When two packets request the same channel at the same time, two questions arise: (1) which packet receives the channel?, and (2) what happens to the other packet? Figure 2.5 shows four common flow control disciplines for dealing with the "other" packet [26].

- ***Buffering (Figure 2.5(a)):*** One packet is granted the channel. The other packet is allowed to continue advancing and is stored in a buffer. This scheme has two disadvantages: (1) It requires a large amount of buffer storage – enough for the largest packet; and (2) the buffers must be allocated in an acyclic manner to avoid deadlocks. The required buffer storage is often more than can be built into a router and must be allocated in the processing node's memory, which slows the router down to the memory cycle time. Also, valuable node memory bandwidth is spent on through messages.

- ***Blocking (Figure 2.5(b)):*** The other packet is stopped in place. Blocking idles the resources that are allocated to the blocked packet. Blocking requires very little storage, results in a small, fast communication controller. Blocking flow control with a small number of flit buffers usually gives best performance for a given set of storage and communication resources.

- ***Dropping (Figure 2.5(c)):*** The other packet is dropped (eliminated). It is allowed to continue advancing, but its flits are not stored as they arrive at the node. The information is lost and must be retransmitted. Dropping results in a severe waste of resources, because all of the channel and buffer time used to get the dropped packet to the point of conflict has been wasted. It also

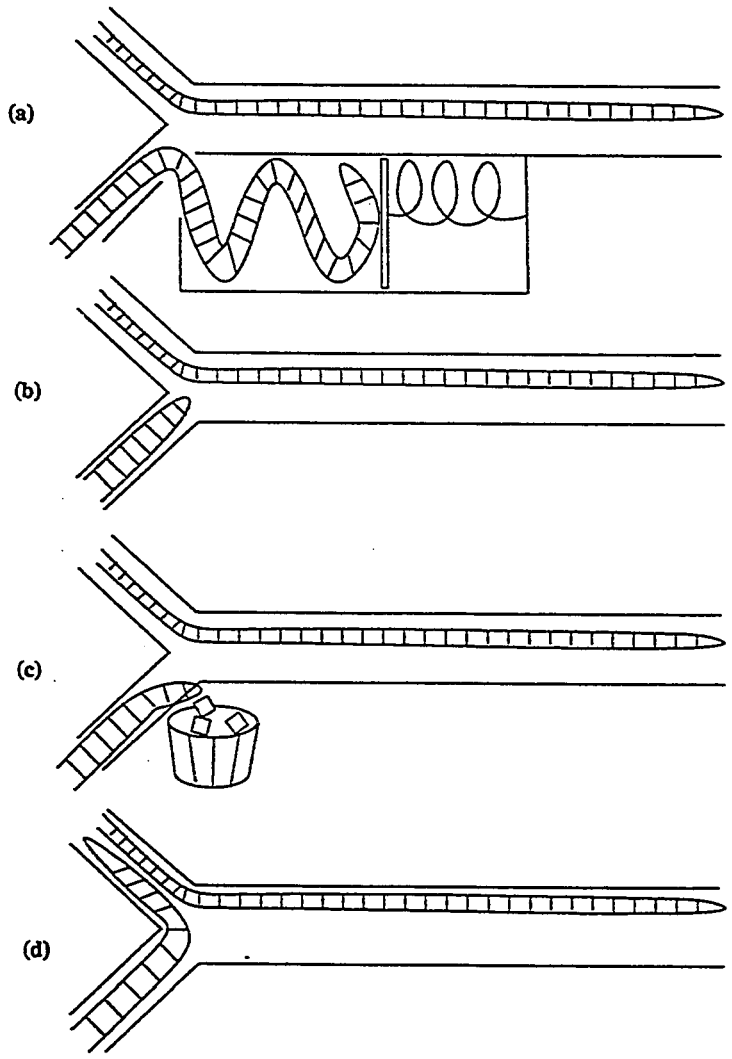


Figure 2.5: Packet collisions may be resolved by (a) buffering (b) blocking (c) dropping (d) misrouting.

has the disadvantage of requiring a positive acknowledgment on packet arrival that further wastes communication resources. In addition, a retransmission protocol is required that wastes computing and storage resources and increases latency with time-outs on dropped packets.

- ***Misrouting (Figure 2.5(d)):*** The other packet is routed to an idle but incorrect channel. This method of resolving packet collisions wastes resources by using channels and buffers to take a packet further from its destination.

Two flow control mechanisms have been recently proposed. These are described as follows:

- ***Drop-and-Reroute***: This is a flow control policy for adaptive wormhole routing proposed in [69]. Drop-and-reroute flow control exploits the adaptivity achieved by adaptive routing allowing a blocked packet to be routed again starting from the preceding node. When a header flit fails to advance to the next node, it is dropped and routing is retried in the previous node from which it was routed. Therefore, if the header flit can release the holding channel (drop) and it can be routed again in one of the preceding nodes (reroute), it may bypass the congestion and thus reduce the packet latency. Drop-and-reroute differs from backtracking in that backtracking is used in the path setup phase of circuit-switching and is not possible with wormhole routing.

- ***Valved Routing***: Proposed by Liao & King [88], ***valved routing*** controls message injection and transmission through logical valves associated with the router ports. These valves (implemented as binary variables) are associated

with the input/output ports of a router and the injection port from the local processor. By setting the valves properly, one can control the message flows between routers and manage accesses to router resources.

Switching

While the input and output selection policies determine how a packet uses channels as it traverses an intermediate router, switching is the actual mechanism that removes data from an input channel and places it on an output channel. Network latency is highly dependent on the switching technique used. Switching techniques will be discussed in detail in Chapter 4.

2.3 Operational Model of Multicomputer Networks

In Figure 2.6 we frame an operational model of a multicomputer network. This model reflects the way the routing task is subdivided into smaller tasks of managing the virtual channels, buffers, and flow control in the physical layer, implementing the switching mechanism and communication primitives in the communication layer and finding the path to the destination in the routing layer. The three layers interact with one another to provide a smooth flow of messages across the interconnection of a multicomputer.

Physical Layer

At the lowest level is the ***physical layer*** and the channel model as depicted

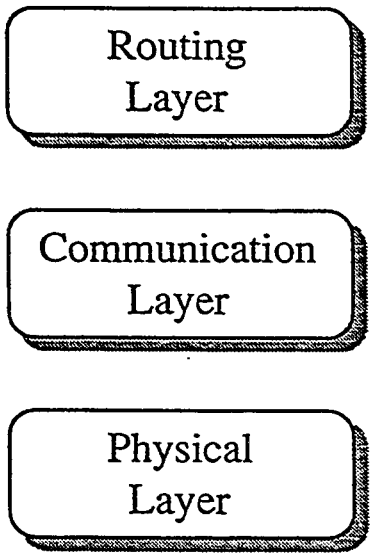


Figure 2.6: Operational Model of Multicomputer Networks.

in Figure 2.7 [98]. Each processing element (PE) in the network is connected to a routing node. The PE and its routing node can operate concurrently. A number of virtual channels are realized in each direction over each physical channel. Each virtual channel is realized by independently managed buffers and they share the physical channel bandwidth. A flow-control mechanism is used to allocate physical channel bandwidth proportionately to virtual channels [48].

Communication Layer

On top of the physical layer is the ***communication layer*** or ***data-link layer*** which consists of two parts: one that implements the switching mechanism and the other where the actual communication protocols are implemented.

Routing Layer

At the topmost level is the ***routing layer***, within the network layer, in which

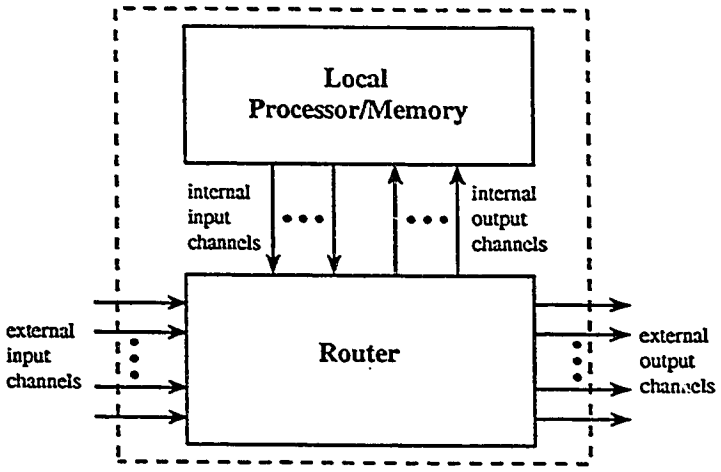


Figure 2.7: Generic node architecture.

routing protocols and their various components are implemented.

2.4 Performance Evaluation Metrics

A primary measure of communication performance is ***traffic*** or ***throughput.*** The ***traffic*** or ***traffic density*** on a link, is the number of messages per time unit using that link. ***Throughput*** is the average number of packets accepted per unit time per node. Two important performance measures derived from traffic are ***link utilization*** and ***message delay.*** The former quantifies the fraction of time a link is used; the latter measures the expected time it takes a message to pass through the network from its source to its destination. ***Mean latency***, is the number of time units spent by a typical packet from its source to its destination, taking the queuing delay into consideration. ***Maximum traffic density*** denotes the longest traffic density among all the links in the system. It is an index as to whether any particular link will carry excessive traffic and could potentially become a communication

bottleneck. Another major performance parameter is ***hop-to-hop time***, which is measured in ***time units***, the actual time needed to send a message from a node to one of its neighboring nodes. This time is assumed to be constant for all pairs of neighboring nodes. The ***maximum internode distance***, often referred to as the ***diameter*** of the interconnection network, is the of communication links traversed to transmit a message to any node along the longest shortest path. It places a lower bound on the delay required to propagate information throughout the network. The ***mean internode distance*** is the expected number of link traversals a typical message needs to reach its destination.

The communication behavior of a system can be characterized by the relationship between mean latency and throughput. For a given system, mean latency generally grows with the increase of throughput. At a low throughput, latency is contributed mainly by the number of hops a typical message makes, because there is little queuing delay (contention) involved. As the throughput increases, more contention and longer queuing delay results, giving rise to higher latency. Near saturation, small increases in throughput cause significant changes in latency .

In the traffic models used to analyze performance of a multicomputer network, often messages are assumed to be generated independently by every node according to a Poisson process. Message transmission times are assumed to be exponentially distributed. Moreover, buffer size of a node is assumed to be infinite as in $M/M/1$ queues. Communication is said to be uniform when each pair of nodes i and j exchange messages at an identical rate.

An important metric built upon some of the above measures is the ***communi-***

cation latency, which is the sum of three component values: ***start-up latency***, ***network latency*** and ***blocking latency***. The ***start-up latency*** refers to the time required for message framing/unframing, memory/buffer copying, validation, and so on, at both source and destination nodes. The startup latency is mainly dependent on the design of a system software within the nodes and the interface between nodes and routers. Startup latency can be further classified into ***sending latency*** and ***receiving latency*** — the start-up latencies incurred at the sending node and the receiving node, respectively.

The ***network latency*** is equal to the elapsed time after the head of a message has entered the network at the source until the tail of the packet emerges at the destination. Given a source and destination node, the startup and network latencies are static values, frequently used to characterize contention-free networks. Start-up latency and network latency are static for a given system, that is the sum of their values reflects the latency of packets sent in the absence of other network traffic and transient system activities.

The ***blocking latency*** includes all possible delays encountered during the life-time of a message. These delays are mainly due to conflicts over the use of shared resources, such as busy channels and filled buffers. Blocking time reflects the dynamic behavior of the network due to passing of multiple messages, and may be high if the network traffic is heavy or unevenly distributed [32], [116].

2.5 Summary

In this chapter we have presented an introduction to multicomputer networks and their various types and characteristics. There are mainly two different types of multicomputer networks, namely ***indirect*** and ***direct***. We will mainly be concerned with the direct networks in the present work. The different characteristics of multicomputer networks described here are topology, routing, flow control, and switching. We also discuss an operational model of multicomputer networks. A discussion of various performance measures used to study the performance of multicomputer networks has also been presented.

Chapter 3

Characterization and Classification of Routing in Multicomputer Networks

Routing is the method used for a message to choose a path over the network channels. Routing can be thought of as involving a relation, R, and a function, ρ.

$$\mathbf{R} \subset C \times N \times C,$$

$$\rho : P(C) \times \alpha \rightarrow C.$$

The routing relation R identifies the permissible paths that may be used by a message to reach its destination. Given the present position of a message, $C_1 \in C$, and its destination node, R identifies a set of permissible channels, $C' \subseteq C$, that can be used as the next step on the route. R is a relation rather than a function because

there may be more than one permissible path for a message to follow.

The function ρ selects one path from the set of permissible paths. At each step of the route, ρ takes the set of possible next channels, $P(C)$, some additional information about the state of the network, α, and chooses a particular channel, C_i. The additional information, α, may be constant, random, or state information based on traffic in the network. This routing function considers the current position of a message to be a channel rather than a node. R is $C \times N \times C$ rather than $N \times N \times C$. Considering the present position of a message to be a channel allows to construct deadlock-free routing relations. If position is considered to be a node, deadlock must be avoided by storage allocation.

The properties of the channels are usually assumed to be the following [86]:

1. The channels connected to a processor operate independently.

2. They are mutually exclusive, i.e., two messages cannot share a channel at the same time.

3. They are non-preemptive, i.e., a message occupying a channel will not relinquish the channel until the message is completely transmitted through the channel.

The allocation of channels and their associated buffers to packets can be viewed from two perspectives. The routing algorithm determines which output channel is selected for a packet arriving on a given input channel. Therefore, routing can be referred to as the ***output selection policy.*** Since an outgoing channel can be requested by packets arriving on many input channels, an ***input selection policy***

is needed to determine which packet may use the output channel. Possible input selection policies include round robin, fixed channel priority, and first-come first-served. The input selection policy affects the fairness of routing algorithms.

Now, we characterize routing and provide classifications for the various types of routing as shown in Figure 3.1.

3.1 Various Types of Routing

One of the most desirable properties of routing protocols is their adaptivity to network conditions. A routing protocol is thus ***adaptive*** or ***dynamic*** if, for a given source and destination, the path taken by a particular packet depends on dynamic network conditions, such as the presence of faults and/or congestion. Thus multiple alternative paths may be used to route messages between two nodes, making efficient use of network bandwidth. As opposed to adaptive, ***deterministic***, ***oblivious***, ***fixed-path***, ***fixed***, ***static***, or ***non-adaptive*** routing is a method in which the path is completely determined by the source and destination addresses. Thus an oblivious protocol uses a single fixed path, even when multiple paths are available, for communication between any source-destination pair in the network, having a choice at every node of only one node to forward each message to, ignoring the network status. Oblivious protocols are generally simple and deadlock-free, but they do not make use of all the bandwidth available between each source-destination pair.

Some of the routing algorithms use a source of randomness to make routing decisions. These algorithms are called ***randomized routing*** algorithms. Algorithms that do not use any kind of randomness are called ***deterministic*** algorithms. In ***di-***

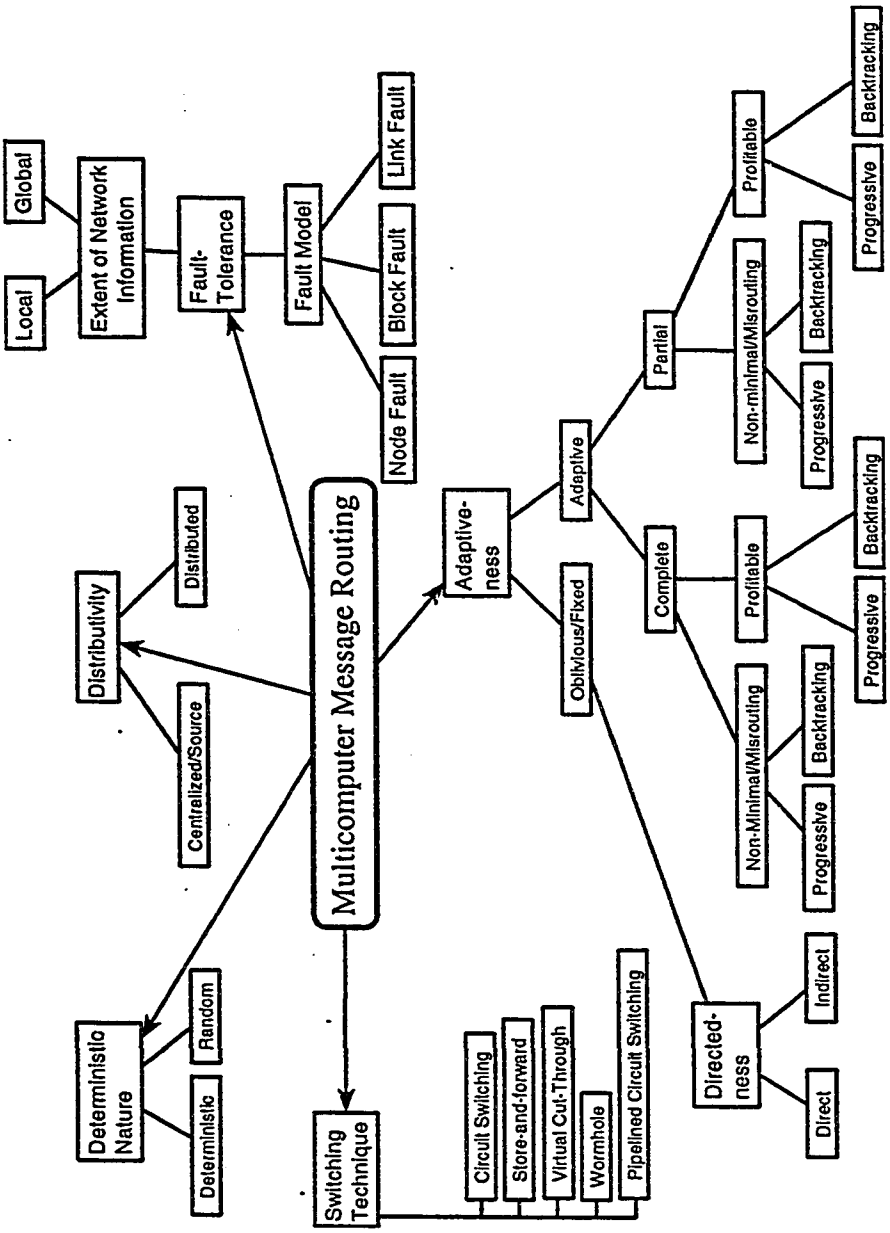


Figure 3.1: A characterization and classification of various routing techniques.

rect routing a message proceeds directly to its final destination according to some specified routing algorithm. With ***indirect routing***, the message uses the specified routing algorithm to first go from its source to some intermediate destination and then to go from the intermediate destination to its final destination. That is, once the message reaches its intermediate destination, it immediately proceeds towards its final destination. Indirect routing appears unnatural as it can cause messages to traverse longer paths than those required by direct routing. However, indirect routing has been proven useful in avoiding congestion [43].

Routing algorithms may be either ***centralized*** (***global control***) or ***distributed*** (***self routing***). In centralized routing, routing decisions regarding the entire path are made before sending a message, solely by a single processor (perhaps the source node in which case the routing is called ***source routing***). Centralized routing requires complete information concerning the rest of the nodes [46].

Source routing is useful in reconfigurable networks. If it is possible for the network topology to change (due to faults or other reasons), the router must be flexible or programmable to allow for the implementation of different deadlock-free routing algorithms. Depending on the underlying network topology, the source node specifies the routing path on the basis of a deadlock-free oblivious routing algorithm. The message header must carry complete routing information [107].

In distributed routing each node decides by itself on which channel to send or forward a message. Thus only information about the neighborhood is needed. In distributed routing, each node, upon receiving a message, invokes a routing algorithm to determine whether the message belongs to it or must be forwarded towards its

destination [46]. Most routing protocols and algorithms developed are distributed in nature.

3.2 Adaptive Routing

Adaptive routing protocols can be classified into 3 different levels as suggested in [46]. At the top level the techniques are either classified as minimal or nonminimal. At the second level they are either backtracking or progressive, and at the lowest level they are completely or partially adaptive. However we have shifted the third level to the top level and the top level to the bottom. We feel this gives a better classification and is more general than the one presented in [46] because when an adaptive routing technique is considered the most important feature is its degree of adaptivity which is reflected by it being completely or partially adaptive. So this could be used as a first criterion to evaluate a routing algorithm that is claimed to be adaptive.

3.2.1 First Level

At the highest level, protocols can be either ***completely (fully)*** or ***partially*** adaptive. A completely adaptive protocol can use all paths in its class (if profitable then all profitable links). It is not deterred by routing restrictions from establishing any profitable path. A backtracking protocol that is completely adaptive is also called ***exhaustive.*** If a protocol is restricted to using a subset of the paths available, then it is said to be ***partially*** adaptive.

All fully adaptive routing algorithms allow a message to use any physical chan-

nel that is part of a shortest path. Full adaptivity is a feature from which one can hope to obtain the best possible performance.

3.2.2 Second Level

At this level protocols are classified as ***misrouting*** (***non-minimal***) or ***profitable*** (***shortest-path***, ***productive***, or ***minimal***), based on the set of links that are candidates for routing at each intermediate node.

A "profitable" protocol only considers profitable links as viable candidates for routing at each node. A ***profitable links*** is a link over which a message moves closer to its destination. A "misrouting" protocol considers both profitable and nonprofitable links as candidates for routing decisions at each node, leading messages to follow a longer than needed path, usually in response to current network conditions. A misrouting protocol may however create an undesirable situation in which a message will continue to be routed through the network but never reach its destination.

Profitable and misrouting protocols hold a conservative view that routing must be carried out only over links known to be guaranteed to move towards the destination. If deadlock avoidance techniques are used, profitable protocols can guarantee to eventually find a path in a fault-free network. However, these are less tolerant to faults. Every faulty link hinders direct communication between at least one pair of processors. Several faulty links can separate large parts of the network. Profitable protocols realize minimal-length paths, and are free from deadlock.

Misrouting techniques are based on an optimistic view of the state of the network

that following a nonprofitable link is likely to bring the search to another set of free profitable links that will allow further progress to the destination. The ability to use nonprofitable links gives misrouting protocols more flexibility and greatly increases the number of possible paths from a source to a destination. This class is perhaps the largest in the taxonomy [46].

3.2.3 Third Level

In this level, adaptive protocols are classified based on whether they are ***progressive*** or ***backtracking***. "Progressive protocols", as the name suggests, move forward and have only limited ability to backup from a node once that node is reached. "Backtracking protocols", on the other hand, search the network systematically, backtracking as necessary. They store history information in the header and use it, when needed, in order to ensure that no path is searched more than once.

Backtracking protocols work on the premise that it is better to be searching for other paths than to be waiting for one to become available. They search the network for a path in a depth-first manner. Most backtracking protocols use the routing header to store history information. Neither deadlock nor livelock is a problem with backtracking protocols because they do not block, holding resources, and they can use history information to avoid searching the same path repeatedly. The use of routing header significantly increases the size of the header over progressive protocols and, consequently, increases the time required to route the probe through the network. This is particularly a problem for backtracking protocols, since the number of links traversed during path setup can be very high. To overcome this

problem the history information can be distributed throughout the nodes of the network, reducing the header size.

To highlight once again, adaptive routing has two advantages:

(1) Congestion can be diffused by exploiting alternative paths to a destination.

(2) If any faulty component is encountered along one path, another path can be taken to preserve communication.

3.3 Routing for Broadcasting and Multicasting

In order to support the operation of massively parallel computer systems, there is a need to provide fast communication mechanisms that may be used to distribute the load among the various processors or for the processors to communicate the results of computations. There are two types of communication operations, one in which a single node is allowed to communicate a message to only a single destination which may be its immediate neighbors or others, called ***unicast*** or ***one-to-one*** or ***point-to-point***; the second is a large class of various operations whereby either multiple sources or multiple destinations or both are permitted, called ***collective communication.*** The most important operations defined in this set are ***broadcast*** and ***multicast.*** In ***broadcast***, one node sends the same message to all other group members. ***Multicast*** refers to a type of communication operation in which one source wants to send its messages to k distinct destinations.

Broadcast is a very common operation needed in a parallel system to distribute code and data from a host processor to a set of node processors before computation

begins. This operation is also used to distribute data from one processor to others during the computational phase of a distributed memory program. In the literature, it is also known as ***one-to-all*** operation with ***non-personalized*** data movement. If multiple broadcasts occur simultaneously, it is identified by ***many-to-all*** operation. The extreme case is ***all-to-all*** where everyone does a broadcast to all others [108].

Multicast is sometimes referred to as a subset operation of broadcast, whereas some authors also refer to broadcast as a special case of multicast. Whatever the case may be, multicast is mainly known as ***one-to-many*** with ***non-personalized*** data movement. In any parallel system, a broadcast operation results in a multicast operation if not all processors participate in the operation. Depending on the number of participating sources, this operation can be identified as ***single multicast*** or ***multiple multicast***. Multicasting involves a source node S and a set of k destination nodes D_i, $1 \leq i \leq k$. The objective of the multicast operation is to deliver a common message from S to each one of the destination nodes.

Multicast communication has several uses in large-scale multiprocessors. First, a growing number of parallel applications, including parallel search algorithms and parallel graph algorithms, have been shown to benefit from the use of multicast services. Second, multicast is useful in the SPMD (Single Program Multiple Data) model of computation, in which the same program is executed on different processors with different data. Specifically, multicast is fundamental to several operations, such as replication and barrier synchronization, supported in data parallel languages. Third, if a distributed shared-memory paradigm is supported, then multicast services may be used to efficiently support shared-data validation and updating [87].

Providing support for multicast involves often conflicting requirements. First, it is desirable that the message delay from the source to each of the destinations be as small as possible. Sending a separate copy of the message to each destination along shortest paths is one solution, but the increased traffic load resulting from these copies may hinder the progress of messages. Hence arises the second requirement that the amount of network traffic be minimized. Unfortunately, finding optimal multicast routes, in terms of delay and traffic, has been shown to be NP-Hard for most multicomputer topologies [91]. The third requirement, then, is that the routing algorithm not be computationally complex. Heuristic algorithms must be used, but are constrained by the final requirement, which stipulate that the multicast communication protocol must be deadlock-free.

Design of multicast protocols and routing algorithms to meet these requirements depends on the topology of the network used to interconnect nodes in the multicomputer. In addition to the network topology, the design and resulting performance of multicast communication protocols also depend on the underlying switching mechanism used in a multicomputer [90].

Practical broadcast and multicast routing algorithms must be deadlock-free and should transmit the message to each destination node in as little time and using as small a path as possible. One approach to this problem is to deliver the message along a common path to as many destinations as possible, and then replicate the message and forward each copy on a different channel band for a unique set of destination nodes. The path followed by each copy may further branch in this manner until the message is delivered to every destination node. In such ***tree-based***

model, the destination set is partitioned at the source, and separate copies are sent on one or more outgoing links. A message may be replicated at intermediate nodes and forwarded along multiple outgoing links towards disjoint subsets of destinations [62].

The tree-based multicast, however, suffers from several drawbacks especially if used in multicomputers that use wormhole routing. Since there is no message buffering at routers, if one branch of the tree is blocked, then all are blocked. Branches must proceed forward in lockstep, which may cause a message to hold many channels for extended periods, thereby increasing network contention. Blockage of any branch of the tree can prevent delivery of the message even to those destination nodes to which paths have successfully been established. Moreover, deadlock can occur using such a routing scheme.

Another way to implement the multicast operation is using ***separate addressing***, in which a separate copy of the message is sent directly from the source to every destination. This method however is very inefficient.

The ***port model*** of a system refers to the number of internal channels at each node. If each node possesses exactly one pair of internal channels, then the architecture is called ***one-port***. A major consequence of one-port architecture is that a node must transmit (receive) messages sequentially. Architectures with multiple ports reduce this bottleneck. An ***all-port*** system is one in which the node can send and receive on all channels simultaneously. In an ***n-port*** architecture, the number of ports is greater than one but less than the number of all the channels.

A characterization of routing techniques for broadcast and multicast communi-

cation is shown in Figure 3.2.

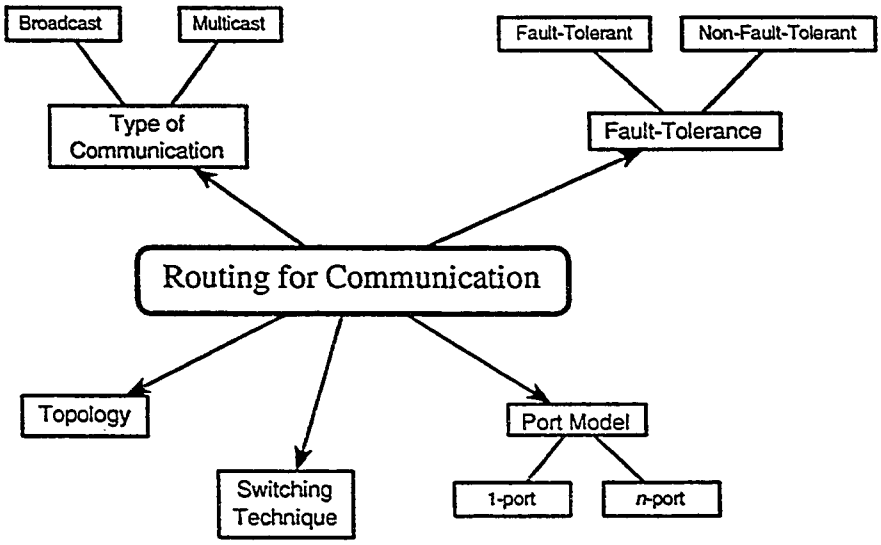


Figure 3.2: A characterization of routing techniques for communication.

Broadcast/Multicast Trees

Many algorithms for collective operations are based on a tree of point-to-point, or *unicast* messages. A *broadcast tree* is defined as one that involves all nodes in the network, and a *multicast tree* as one that involves a subset of the nodes, or if the operation involves only a subset of the application processes. Multicast trees are needed if the application is allocated only a subset of the nodes, or if the operation involves only a subset of the application processes. Although a broadcast operation can be implemented with a multicast tree algorithm, this special case is often treated separately since the participation of all nodes leads to simpler and more regular message-passing patterns [97].

Early multicomputers used store-and-forward switching, so many early tree-based algorithms use only nearest-neighbor communication. The simplest broad-

cast tree is a Hamiltonian path. Figure 3.3(a) illustrates one such path in a 4-dimensional hypercube. The path, which begins at node 0000 and ends at node 1000, is based on a binary-reflected Gray code. In either a store-and-forward or a wormhole routed network, the time needed to complete a broadcast operation in this manner is $(N-1) \times t_u$, where N is the number of nodes in the network and t_u is the time required to send a unicast message [97].

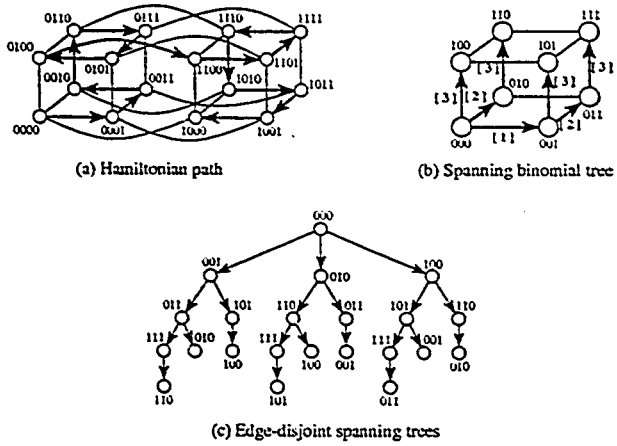


Figure 3.3: Hypercube broadcast trees based on nearest-neighbor communication.

An approach that makes better use of the dense hypercube topology is the ***spanning binomial tree*** (SBT) algorithm [129], which is illustrated for a 3-dimensional hypercube in Figure 3.3(b). The steps of the algorithm are indicated in brackets. In the first step, the source node sends the message to its neighbor whose address differs from its own in the lowest bit position. In the second step, these two nodes send the message to their respective neighbors in the next dimension. This ***recursive doubling*** process continues until, in the last step, half of the nodes in the hypercube forward the message to the other half through the highest dimension. The algorithm requires n message-passing steps to reach all nodes in an n-dimensional hypercube

[97].

A technique that is often used to reduce the execution time of communication operations is ***message partitioning***. An example of this is the ***edge-disjoint spanning tree*** (EDST) broadcast algorithm [62] for hypercubes. In this approach, the message is partitioned into n segments, each of which is transmitted along a different spanning tree, as illustrated in Figure 3.3(c). By spreading message segments among more channels than the SBT algorithm, the EDST algorithm completes the broadcast of a message of length l in time $O(l/n)log_2N) = O(l)$ [97]. Since the spanning trees are disjoint, this algorithm does not incur channel contention. However, the algorithm does require that the message be reconstructed at every destination. Moreover, if the architecture does not support a sufficient number of input and output ports, then this approach can incur contention at the ports. For example, in Figure 3.3(c), node 111 may be required to receive three message segments at approximately the same time.

3.4 Potential Problems Encountered in Routing

Routing has its own difficulties and problems which must be addressed and solved before a technique could work properly. Some of these difficulties are the occurrence of deadlock, livelock and starvation. The designer of a routing scheme must at least guarantee that the proposed technique avoids these problems or contains a mechanism to handle them when one of these occurs inadvertently. This is so because if one of these problems exists in a section of an multicomputer network, it will affect the entire network paralyzing it completely or at least rendering parts of it unusable.

We now describe these problems in detail, explaining what leads to such situations and what could be the possible solutions.

3.4.1 Deadlock

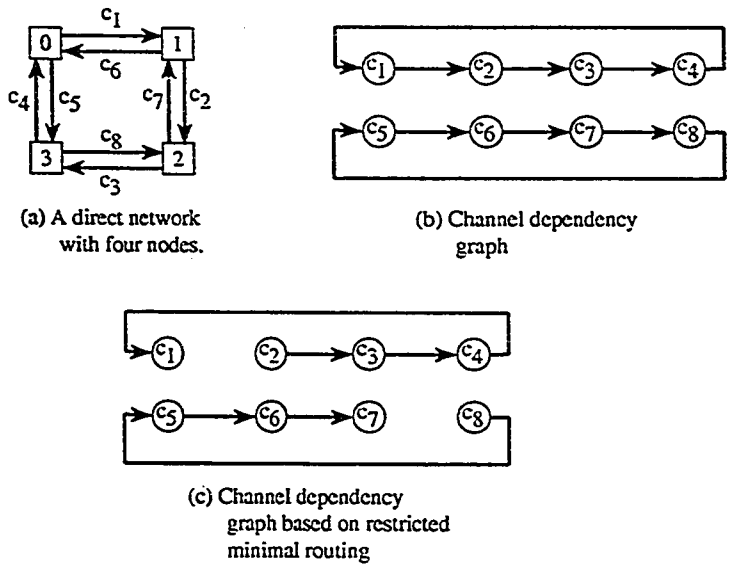
There are three highly undesirable characteristics of routing techniques which should be avoided altogether in any reliable routing algorithm. The first of these is ***deadlock*** which results from cyclic dependencies within the network. Deadlock can occur if packets are allowed to hold some resources while requesting others. Guaranteeing deadlock freedom is a problem when designing a routing technique. Every algorithm must have either some mechanism in order to avoid deadlock or a way to recover from it if it occurs [42].

One way to solve the deadlock problem is to allow the preemption of packets involved in a potential deadlock situation. Preempted packets can be either rerouted or discarded. The former policy gives rise to adaptive nonminimal routing techniques. The latter policy requires that the packets be recovered at the source and retransmitted. Because of requirements for low latency and high reliability, packet preemption is not used in most direct network architectures.

More commonly, deadlock is avoided by the routing algorithm. By ordering network resources and requiring that packets request and use these resources in strictly monotonic order, circular wait — a necessary condition for deadlock — is avoided. Hence, deadlock involving these resources cannot arise.

In wormhole-routed networks, channels are the critical resources. A ***channel dependency graph*** can be used to develop a deadlock-free routing algorithm. The

channel dependency graph is a directed graph $D = G(C, E)$, where the vertex set $C(D)$ consists of all the unidirectional channels in the network, and the edge set $E(D)$ includes all the pairs of connected channels, as defined by the routing algorithm. In other words, if $(c_i, c_j) \in E(D)$, then c_i and c_j are, respectively, an input channel and an output channel of a node, and the routing algorithm may route packets from c_i to c_j. A routing algorithm for a direct network is deadlock-free if and only if there is no cycle in the channel dependency graph [107].



(a) A direct network with four nodes.

(b) Channel dependency graph

(c) Channel dependency graph based on restricted minimal routing

Figure 3.4: A four-node network and the corresponding channel dependency graphs.

Figure 3.4 demonstrates the channel dependency graph method. The four nodes shown in Figure 3.4(a) can be considered as a ring, a 2×2 mesh, a 2-cube, a 4-ary 1-cube, or a 2×2 torus. Assuming a packet can be delivered through any minimal routing path, the corresponding channel dependency graph is shown in Figure 3.4(b). Since there are two cycles in the channel dependency graph, deadlock is possible. One way to avoid deadlock is to disallow packets to be forwarded from channel c_1

to c_2 and from c_7 to c_8. The resulting channel dependency graph is shown in Figure 3.4(c) . It can easily be verified that the routing is still minimal. However, to send a packet from node 0 to node 2, the packet must be forwarded through node 3, as the path through node 1 is no longer permitted.

With store-and-forward or cut-through switching, storage resources are allocated entirely in terms of packets. Once the first flit of a packet is accepted, sufficient storage must be available to hold the entire packet. A method of breaking dependency among packet buffers is to structure the buffer pools so they form a partial order. Packet buffers are associated with nodes, so once a packet is buffered there is no history of the channel it arrived on. The routing relation is $N \times N \times C$ [26].

The buffer pool is structured so that a packet in a given buffer, A, can be stored only in a restricted set of buffers, S, on the next node of the route. If buffer $B \in S$, then there is a dependency from A to B, $dep(B, A)$. Buffer allocation is restricted so the buffer dependency graph, the directed graph defined by the dependency function, *dep*, is acyclic.

For example, allocating buffers to packets depending on the distance from their destination node results in an acyclic buffer dependency graph. If the diameter of a network is D_{max}, then D_{max} packet buffers can be created on each node. At each step of a route, if a packet is h hops from its destination, it can route only into the h^{th} buffer on the next node of the route.

Although structured buffer pool deadlock avoidance works well for local area and wide area networks, the large amounts of storage required and the high latency make it inappropriate for multicomputers [26].

3.4.2 Livelock

Messages do not have to wait for busy resources, but can be misrouted in hope of finding another path to the destination. This behavior can lead to another situation known as ***livelock***. Livelock occurs when a message circulates endlessly in the network, never reaching its destination. Algorithms that misroute packets in such a fashion must have some way of dealing with livelock [46]. When processors want to communicate, they ***inject*** their messages in the network. If all of them do so at the same moment, with the network clear of messages, the ***injection model*** is called ***static***. If every processor can inject messages at an arbitrary time, the injection model is called ***dynamic***. Livelock cannot take place in a static injection model if the queuing policy is ***fair*** and the routes are minimal. The first condition means that no message can be forever prevented from routing if it can be routed. All sensible queuing policies are fair, so livelock is not a problem when injection is static in minimal routing techniques. The real problem appears with dynamic injection. Imagine a routing algorithm that routes first those messages that are closest to their destination. Furthermore, suppose that node n has a message m that is at distance 2 from its destination. If n injects an infinite sequence of messages with distance 1 to their destination, m will wait forever in the queue. In nonminimal routing, livelock can also take place if a message can be derouted forever [42].

Different policies exist for avoiding livelock. In general, they are based upon the following. Let P be a set of priorities which is totally ordered. Whenever a packet is injected into the network, some priority is assigned to it. It must hold that:

(a) packets are routed according to their priorities.

(b) once a packet has been injected, only a ***finite*** number of packets will be injected with a higher or equal priority.

A related policy is one that assigns every packet some minimum priority when injected and increases it as the packet remains undelivered. Both of these policies are actually the same. These policies have been criticized because they have at least two drawbacks. The first one is the fact that priority queues must be implemented to maintain the packets ordered, and such queues are more expensive and slower than the simple FIFO queues. The second drawback is the need for more bits in the packets in order to store the priorities. It is also costly to update them [42].

3.4.3 Starvation

A node suffers from ***starvation*** when it has a packet to inject in the network and it is never allowed to do so. Starvation cannot arise if the injection model is static. The way starvation is avoided is related to the injection control policy. The goal is to assure that every node can eventually inject packets into the network. The main policies proposed in order to avoid starvation are the following [42].

a. **Injection Completion:** Every node has an injection queue, where it stores the messages it wants to inject into the network. This queue is considered in the same way as the queues of the incoming links to that node, and it competes with them. As the queue management policy is fair, this method avoids starvation. Its main advantage is also its main drawback: its simplicity. This is so because a node with a high injection rate can slow down all the others in the network.

b. **Private-Buffer Recirculation:** This policy allows every node to reserve some fraction of its own internal message queue for injection. Each packet must have a mark saying if it was injected in the private fraction or in the public one. When a packet injected in the internal fraction of its sender is delivered to its final destination, the receiver uses the buffer used by the delivered packet to inject a ***null*** message destined to the sender. When a node needs to inject a message in the network, it looks for a free buffer in its queue. If it finds one, the packet is injected there. Otherwise, the node must wait for the return of some ***null*** packet sent to it, and then, it uses that buffer to inject its packet. This policy has been criticized because it makes the number of circulating messages with packets that have no information too large.

c. **Injection-Token Recirculation:** This policy has been proposed as an alternative to the above policy. A function $Next : V \rightarrow V$ must be built defining a cycle involving all the nodes of the network. There exists a number of ***token*** packets in the network, with null content and a destination. When a token packet arrives at its destination node n, this node determines whether it has been prevented from injecting for longer than a fixed quantum q. If it has and it wants to inject, then it uses the buffer occupied by the token to inject its packet. Otherwise, it sends the token packet to $Next(n)$. Whenever a packet arrives at its final destination, the receiver must check if the message has been injected over a token packet. If it has, then the receiver must regenerate the token packet, with destination equal to the $Next$ node of the packet's sender.

This policy can be generalized letting $Next$ define many disjoint cycles covering all the nodes of the network instead of just one with at least one token packet in every cycle. The Private-Buffer Recirculation policy can be thought of as a particular instance of this policy, when $Next$ is the identity permutation and $q_n = 0$ for all n in the network.

d. **Packet-Injection Control:** It is based on putting a bound on the difference between a node's injection rate and that of its neighbors. This is done via some kind of information exchanged between a node and its neighbors. A node is allowed to inject if the difference between the number of packets it has injected itself and the number of packets injected by its neighbors is greater than $-k$ for all its neighbors, for some fixed k. This policy is more adaptive to changing injection rates than the previous ones and does not have the drawback of augmenting the number of circulating packets in the network. As a disadvantage, it is complex and its implementation may be expensive.

3.5 Summary

In this chapter we have proposed a classification and characterization scheme for routing techniques in multicomputer networks. Various properties of the class of adaptive routing techniques have been described in detail. A discussion of the use of routing for broadcast and multicast communication has also been conducted. Different techniques for constructing the broadcast and multicast trees have also been outlined. A description of some of the problems encountered in routing and sugges-

tions to their solution has also been presented. In summary the most demanding requirements for a routing mechanism are the following [61]:

- the routing protocol must be deadlock free;
- no message is infinitely delayed in the network;
- a packet always takes the shortest route to its destination;
- the routing mechanism must adapt to traffic conditions and exploit the full available communications bandwidth;
- the highest possible throughput must be achieved;
- the lowest possible latency must be achieved.

Chapter 4

Switching Techniques and Virtual Channels

While the input and output selection policies determine how a packet uses channels as it traverses an intermediate router, switching is the actual mechanism that removes data from an input channel and places it on an output channel. Network latency is highly dependent on the switching technique used. A number of switching techniques have been lately proposed for multicomputer networks, of which we studied the store-and-forward, circuit-switching, virtual cut-through, wormhole routing, and pipelined circuit-switching. Some other techniques are also presented which are not so widely used in practical multicomputer systems, but do carry useful ideas.

4.1 Store-and-Forward

Earlier direct networks used ***store-and-forward switching***, also called ***packet switching***, borrowed from computer networks. In this approach, when a packet reaches an intermediate node, the entire packet is stored in a packet buffer. The packet is then forwarded to a selected neighboring node when the next output channel is available and the neighboring node has an unused buffer. In this approach, messages are divided into fixed, hopefully optimal, sized packets (meaning that the packet size has an upper bound), and those packets are thus sent using store-and-forward mechanism. An extreme case of store-and-forward switching is called ***message switching*** whereby the whole message in its entirety is transmitted at each node thus requiring very large buffers and tying up the interconnection for very long period, which is effectively useless for interactive traffic [130].

In store-and-forward switching nonetheless, there is no need for repeated attempts at establishing a connection when the network is under load and most of the resources of the networks are busy. The network in this case accepts the message and undertakes to deliver it whenever this becomes possible. In this type of network, the increase in overall load can result in congestion but the effect may not be felt immediately as the system continues to accept messages for some time. But if the load continues to increase, then the buffers can be exhausted and no more messages will be accepted. This situation continues until at least one of the held up messages is forwarded. There is potentially a catastrophic problem in store-and-forward networks which is called the ***store-and-forward deadlock***. This deadlock occurs when messages traveling in the communication system develop dependency loops

among themselves. This prevents further movement of the messages. Possibility of deadlock arises when a message must travel through intermediate nodes. Because of this, some portions of the message must always be stored at these nodes. Since the storage buffers are finite, they may become filled. When this occurs new messages cannot enter the node until the old messages start to leave. In other words, this happens when a set of buffers exists such that each buffer within that set contains a packet that is destined for another buffer in the same set. This deadlock can be ***direct*** or ***indirect***. The situation is depicted in Figure 4.1.

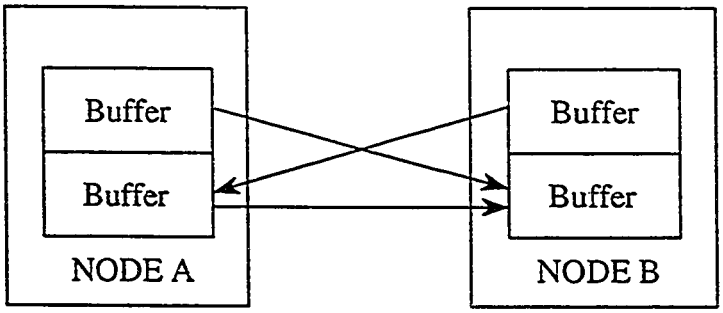


Figure 4.1: A deadlock situation.

These deadlocks may be resolved using buffer reservation (BR) schemes [24]. A buffer-reservation algorithm consists of rules that specify to which buffer a packet may move based solely on the buffer currently holding the packet, the packet's source node, and the packet's destination node. A packet is allowed to move from its current buffer to any other buffer at any time, provided that the other buffer is empty and that the move is allowed by the routing algorithm. The BR schemes are local in nature and can be implemented very easily without the need for complex global synchronization mechanism. A BR scheme provides a pool of buffers which is properly structured, but the number of buffers necessary to maintain this structure

tends to grow with the size of the network. This is not desirable for a computer with thousands of nodes.

Deadlock can also be avoided by using virtual channels, i.e., by multiplexing several logical channels over a single physical channel. Deadlock can thus be avoided by imposing routing restrictions on virtual channels to break cyclic dependencies.

4.2 Circuit Switching

In ***circuit switching***, also called ***path lockdown***, a physical circuit is constructed between the source and destination nodes during the circuit establishment phase. In the packet transmission phase, the packet is transmitted along the circuit to the destination. During this phase, the channels constituting the circuit are reserved exclusively for the circuit, hence, there is no need for buffers at the intermediate nodes. In circuit termination phase, the circuit is torn down as the tail of the packet is transmitted.

The network latency for circuit switching is $(L_c/B)D + L/B$, where L_c is the length of the ***control packet*** (***header***) transmitted to establish the circuit. If $L_c \ll L$, the distance D has a negligible effect on the network latency. If a circuit cannot be established because a desired channel is being used by other packets, the circuit is said to be ***blocked***. Depending on the way blocked circuits are handled, the partial circuit may be torn down, with establishment to be attempted later. This policy of tearing down the circuit is called ***loss mode*** [107].

A commonly used policy to establish a circuit-switched path between a source and a destination is the ***reserve-and-hold*** policy as per which the establishment

of the path is achieved by reserving one link at a time. The path establishment halts at an intermediate node, if the next link along the path is reserved for another source-destination pair.

4.3 Virtual Cut-Through

In order to decrease the amount of time spent transmitting data, Kermani & Kleinrock [64] introduced the ***virtual cut-through*** method for computer communication networks. In virtual cut-through, a packet is stored at an intermediate node only if the next required channel is busy. The network latency is $(L_h/B)D + L/B$, where L_h is the length of the header field. When $L \gg L_h$, the second term, L/B will dominate, and the distance D will produce a negligible effect on the network latency [107]. With the current VLSI technology, it is now possible to implement this switching scheme for multicomputer interconnection networks, and example implementations of it are the network controllers like the Torus routing chip and the HARTS routing controller [63].

This switching system is very similar to packet switching, with the difference that when a message arrives at an intermediate node and its selected outgoing channel is free (just after the reception of the header), then, in contrast to packet switching, the message is sent out to the adjacent node towards its destination before it is received completely at the node. Therefore, the delay due to unnecessary buffering in front of an idle channel is avoided.

The operation of advancing a message immediately from incoming to outgoing channels is referred to as *cut-through*. Because the message can be sent out

immediately after its header (rather than the entire message) is received, we say a ***cut-through*** has occurred. If, after the reception of the header, the message cannot be sent out immediately (due to busy channels), then it must be received completely before being transmitted out (i.e., partial cuts are not allowed) [64].

4.4 Wormhole Routing

Wormhole routing also uses a cut-through approach to switching. A packet is divided into a number of ***flits*** (***fl***ow control dig***its***) for transmission. The header flit (or flits) govern(s) the route. Wormhole routing is an utterly simple idea. The essence of it is depicted in a rather cartoon like Figure 4.2 [123].

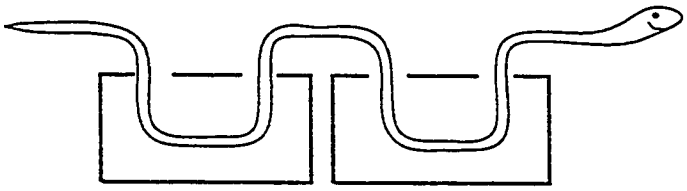


Figure 4.2: The idea of wormhole routing.

It is particularly popular because it has a simple hardware implementation that can provide for high-bandwidth communications, and also because there is little added delay for every hop a message needs to take under contention-free conditions. The second factor means that message latency is virtually independent of the number of hops that the message needs to travel if there were no contention for the communication channels. Unfortunately, wormhole routing has the blocking property which is a characteristic of circuit-switched networks. Consequently, its performance degrades rapidly as the message traffic increases beyond some level as

a high percentage of the messages get blocked before they can reach their destination. The other drawback of a blocking technique is the possibility of deadlock if the routing scheme is not appropriately constrained [35].

In realizing wormhole switching, a small buffer called the ***flit buffer*** is associated with each channel to store a flit. As the header advances along the specified route, the remaining flits follow in a pipeline fashion. Because most flits contain no routing information, the flits in a message must remain in contiguous channels of the network and cannot be interleaved with the flits of other messages [30]. Once one flit of a message is accepted and assigned a buffer, the remaining flits must be accepted by the channel. If the header flit encounters a channel already in use, it is blocked until the channel becomes free. Rather than buffering the remaining flits by removing them from the network channels, as in virtual cut-through, the flow control within the network blocks the trailing flits and they remain in flit buffers along the established route. The network latency for wormhole routing is $(L_f/B)D + L/B$, where L_f is the length of each flit. If $L_f \ll L$, the path length D will not significantly affect the network latency unless it is very large [107]. If there is no blocked communication link between the source and destination nodes, the message is routed to the destination as in a circuit switching network.

Virtual cut-through and wormhole routing are collectively referred to as ***cut-through (switching) methods.*** The cut-through operation is shown in Figure 4.3. The example illustrates how the header moves forward and the data flits follow. A message of 10 bits originates from node u. At node v, part of the message has been received in a FIFO buffer while the head of the message has already been sent

on an outgoing transmitter. At node w, the leading 3 bits of the message have been received in a FIFO buffer. Before the 5th bit arrives at node w, either an outgoing transmitter must be reserved for the message or node w must be prepared to store/receive the message. Thus, of the 10 bits in the message, the first 3 bits are in node w, the middle 4 bits are in node v, and the last 3 bits are in source node u. It is noted that when a message is being cut-through a node such as node v in Figure 4.3, it is possible for node v to also receive the message by copying the message as it passes the FIFO buffer.

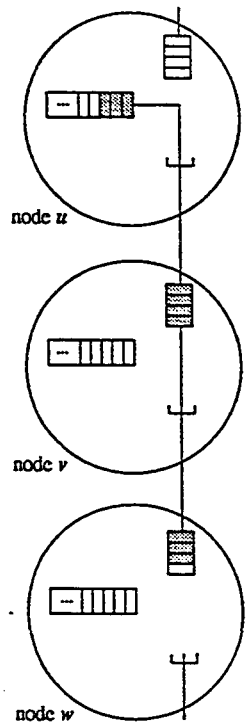


Figure.4.3: Message route diagram.

With wormhole routing the conceptual model for the communications portion of a node is a set of queues-one for each input channel. Instead of one channel dump-

ing a message into a buffer and that buffer being emptied into another channel, the queue of an input channel is connected to an output channel, and the message flits flow through the connection until the entire message has passed. Then the communication can be broken and other connections made. Deadlock can be prevented by restricting the combinations in which the input and output channels are connected. But this has the undesirable effect of restricting the paths that a message can take between a particular pair of nodes. With wormhole routing, we cannot restrict buffer allocation to break deadlock. Instead we must restrict routing.

Many techniques have been developed to reduce the storage requirements of deadlock-free store-and-forward and virtual cut-through routing algorithms. These techniques can be divided into two classes, namely those which require only central queues [25], and those which require that each node have queues (often called *buffers*) that are associated with each edge that is incident to the node.

Algorithms in the first class often require less storage than those in the second class. However, the central queues can become sequential bottlenecks, so algorithms in the second class may offer better performance. In addition, deadlock-free wormhole routing algorithms can be used to obtain deadlock-free store-and-forward and virtual cut-through algorithms in the second class [24].

While circuit-switched and packet-switched mechanisms can be employed by any of the classes of routing protocols, some virtual cut-through routing cannot. If buffers can contain the entire message, virtual cut-through routing can be considered equivalent to packet switching so far as adaptive routing is concerned.

Small buffer schemes such as wormhole routing, are incompatible with back-

tracking protocols, since the pipelined nature of data flow makes it infeasible to backtrack across links. Thus, wormhole routing protocols are restricted to be progressive. Furthermore, since misrouting can cause cycles in the data path that can lead to deadlock, wormhole and small-buffer virtual cut-through protocols are restricted to be profitable or to have some other means of dealing with deadlock (e.g., routing restrictions) [46].

The current state-of-the-art in high speed inter-processor communication is however characterized by adaptive routing in wormhole routed networks using virtual channels and routing restrictions to guarantee deadlock-freedom. Wormhole routing is being used in several multicomputers such as Symult 2010, Ncube2, and iWarp. A detailed performance analysis of wormhole routing is carried out for k-ary n-cubes by Chien [19], Dally [27] and Draper & Ghosh [35] and for hypercubes by Kim & Das [65]. Hady [58] and Agarwal [2] presents an analytical model of network latency for wormhole-routed, high dimension, unidirectional networks with end-around connections. Several other approaches have used the framework of queuing theory [66] [57] to construct models of physical links. A comparison of different wormhole routing schemes has been carried out using simulations in [95].

Several techniques have been suggested and/or implemented which augment the basic wormhole routing scheme to achieve performance improvements while maintaining a simple implementation of a routing element. For example, the links connecting the nodes to routing elements in the iWarp machine have a higher bandwidth than network channels. The purpose of this addition is to remove any bottleneck that is imposed by these channels. The Torus routing chip contains output queues

which have a 4-flit capacity. While only one flit buffer per channel is needed to implement wormhole routing, the output queues allow the time spent in waiting for a channel to be overlapped with the time spent in traversing another channel. Therefore, message latencies are reduced [35].

4.5 Pipelined Circuit-Switching

The ***pipelined circuit-switching*** (PCS) mechanism has recently been proposed by Gaughan & Yalamanchili [18] in order to reconcile the conflicting demands of performance and fault-tolerance in interprocessor communication. PCS is a variant of the wormhole routing mechanism and it relaxes some of the routing constraints imposed by wormhole routing thus enabling routing behavior that cannot otherwise be realized. In fact, PCS makes it possible to realize backtracking misrouting protocols, which wormhole routing does not provide for. The performance model for PCS has been analyzed in [47].

In wormhole routing, header flits containing routing information establish a path through the network from source to destination. Data flits are pipelined through the path immediately following the routing header. Figure 4.4 illustrates a time-space diagram of such a message pipeline over three links in the absence of any other network traffic. The boxes represent the time that each link spends transmitting flits in the message over the link. The white boxes represent the transmission of data flits. The shaded boxes are the transmission of routing header flits. In pipelined circuit-switching, data flits do not immediately follow the header flits into the network. Consequently, increased flexibility is available in routing the header flits. For exam-

ple, rather than blocking on busy channels, the header may "backtrack" and release previously reserved virtual channels on the path and attempt an alternative path to the destination. If physical bandwidth is allocated to virtual channels on a demand-driven basis, the amount of bandwidth used by reserved virtual channels during the setup phase of a PCS circuit is very small. When the header finally reaches the destination node, an ***acknowledgment flit*** returns to the source node. Now data flits can be pipelined over the path just as in wormhole routing. A time-space diagram of a PCS message transmission is shown in Figure 4.4. Shaded boxes in the diagram represent the transmission of routing header flits and acknowledgment flits. The increased routing flexibility of PCS over wormhole routing is obtained at the expense of larger setup times.

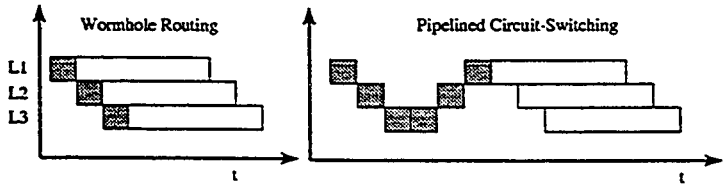


Figure 4.4: Cut-through operation.

If L is the message length in flits and l is the number of links in the path, let one network cycle be the time required to transfer one flit across a routing node and a link, then the minimum latency expression of a message transmitted with wormhole routing is

$$t_{wormhole\ routing} = l + L$$

since it takes l cycles for the header to traverse the path and L cycles for all the

data flits to arrive after that. For PCS we have

$$t_{PCS} = 3l + L - 1$$

since it takes l cycles to deliver the header, l to return the acknowledgment, l to deliver the first data flit and $L - 1$ to deliver the remaining flits. These expressions assume that intranode delays for routing and data flit transmission are comparable.

Routing restrictions on the use of virtual channels can be applied to both wormhole routing and PCS to achieve freedom from deadlock. However, when static faults are present, wormhole routing messages, which must use progressive routing protocols, can block indefinitely. In contrast, routing algorithms can be designed for PCS that can find alternative paths. Since data does not immediately follow the header, when all outgoing links from an intermediate node are faulty, the header may "backup" or backtrack to the previous node on the path releasing a virtual link. Alternative paths out of this previous node may now be pursued. Since headers do not block, holding channel or buffer resources, routing restrictions are not necessary to avoid deadlock. This increases the probability of finding a path while still avoiding deadlocked configurations of messages. Thus, multi-path networks in conjunction with the flexibility of PCS are good candidates for providing low latency fault-tolerant performance. For purely performance-driven applications where fault-tolerance is not a primary concern, the added overhead of PCS makes wormhole routing the mechanism of choice.

PCS acknowledgment flits are returned to the source node via a complementary virtual circuit. Each virtual link v over which data can be transmitted is paired with

a ***complementary virtual channel*** v^* that traverses the same physical link in the opposite direction. (The * operator is symmetric, i.e., $v = (v^*)^*$). The complementary channel v^* is reserved for use by acknowledgment flits, and header flits which are backtracking across channel v (since virtual channels are unidirectional, header flits use the complementary channel). Both header flits and acknowledgment flits are collectively referred to as ***control flits***. v is usually called the ***virtual data channel*** and v^* the ***complementary channel*** or ***virtual control channel***. The pair (v, v^*) is called the ***virtual channel pair***. Virtual channel pairs are statically coupled. When a routing header reserves/ releases a channel v, it also reserves/releases v^*. Hence PCS requires an additional virtual control channel for every virtual data channel that can be used for data transmission. While this increases the complexity of the routing node, the additional cost is found to be considerably less than that of simply doubling the number of virtual data channels. Gaughan & Yalamanchili [47] presented an analytical performance model of pipelined communication in k-ary n-cubes. The model is applicable to both wormhole routing and pipelined circuit-switching.

Extensions to PCS and wormhole routing have been proposed to develop fault-tolerance to dynamic faults such that dynamic fault-recovery is possible. This lead to the development of ***acknowledged pipelined circuit-switching*** (APCS) and ***acknowledged wormhole routing*** (AWR) mechanisms [45]. This dynamic fault-recovery mechanism provides for message abortion and reliable notification of the source and destination nodes of the success/failure of the message transmission. The mechanism does not require time-outs, is fully distributed and prevents deadlock

in the network by removing flits that are blocked on faulty links. In APCS and AWR, when data transmission is complete an acknowledgment is transmitted from destination to the source. This message acknowledgment tears down the path and is necessary to ensure that the source node is informed when message transmission is interrupted by dynamic faults.

4.6 Other Recently Proposed Switching Techniques

We have presented above a description of switching techniques that are in use in practical multicomputer systems and detailed analytical models have been developed for them. However, there are still a number of switching techniques that have received little attention from the practical point of view. Below, we present some such recently proposed switching techniques.

4.6.1 Worm-Through Routing

Worm-through routing [14] adapts a flow control policy somewhat similar to the sliding window protocol. It offers all the benefits of wormhole routing and takes preventive measures to avoid congestion at the expense of some hardware added to the processor-to-network interface. If a sender receives no acknowledgment from its destination after sending some packets, then the sender stops temporarily sending packets to the destination for a while. This leads the sender to reduce packet rate with respect to the destination in an efficient way in the sense that the sender may

start interleaving packets for different destinations. The worm-through routing, as in pipelined circuit-switching, uses a complementary virtual channel for each virtual data channel that is used by the acknowledgments traversing the same physical link in the opposite direction. If the traffic of a network does not cause congestion, the throughput of worm-through routing is the same as that of wormhole routing. But when the network traffic causes congestion and blocking in wormhole routing, the throughput of worm-through routing becomes much higher than the saturation throughput of wormhole routing.

An adaptive version of the worm-through routing is also proposed. When the header flit arrives in a node, the routing table specifies the outgoing links that should be taken to forward the header flit to its destination. If none of the specified outgoing links is available, then the flit are stored in a large size buffer to prevent them from being lost until one of the specified links becomes available.

4.6.2 Scout Routing

Scout routing (SR) [33, 40] is a flow control mechanism that demonstrates that the complete decoupling of path setup and data transmission is unnecessary for tolerating network faults. This provides the header flit with the necessary degree of routing flexibility for controlled backtracking to avoid faults. The attractive property of SR is that with $delay = 0$, it operates as wormhole routing. If the delay is sufficiently large, it operates similar to pipelined circuit-switching.

The use of configurable flow control mechanisms is proposed in [33] where the value of the delay of the first data flit can be set dynamically for individual virtual

channels. Routing protocols can then be designed such that in the vicinity of faulty components messages use SR flow control with controlled misrouting and backtracking to avoid faults and deadlocked configurations. Messages use wormhole routing in fault-free portions of the network. Such protocols are referred to as ***two-phase*** protocols.

4.6.3 Adaptive Virtual Cut-Through

An adaptive version of virtual cut-through is considered in [83] as an alternative to wormhole routing for fast and hardware-efficient interprocessor communications in multicomputers. When originally proposed, virtual cut-through [64] was described as a deterministic routing technique. This model was augmented with adaptiveness to be able to navigate around congested or faulty node/links.

When the header of a packet is received at an incoming channel, the destination node information is extracted from the header. A connection is then attempted to the first-choice outgoing channel. If successful, then a cut-through is established through the current node. However, if the requested outgoing channel is occupied by another packet, then a connection is attempted to the second-choice outgoing channel. This process is repeated until a cut-through is achieved, even using the channel directed backward with preference given to channels directed toward the destination node, or all permissible outgoing channels are exhausted. In the latter case, the packet blocks in the network as in wormhole routing, until one of the permissible outgoing channels becomes available or the waiting time expires. If the waiting time expires the packet is received at the current node and inserted into the

source buffer of the current node just as if the packet originated there. The packet at the head of the source buffer waits until one of the outgoing channels permitted to it becomes available, and then exits through that outgoing channel.

4.6.4 Hot Potato Wormhole Routing

Optical communication is becoming more and more popular, with its bandwidth outperforming electronic communication. However, there is a huge obstacle here: no optical storage which is cheap and fast is available. Moreover, electro-optic conversion at each node of the path, in order to use electro-storage, slows down the potential of optical pulse generation techniques in forming optical flits at high data rates. A way to keep the high communication bandwidth is to use ***hot potato*** wormhole routing. Routing the worms of wormhole routing technique, in the hot potato style is a desired form of communication in high-speed optical interconnection networks [102]. Consider the body of the worm which is instantaneously directed to the outgoing edge. Such a fast switching mechanism is essential for optical communication networks. With hot potato wormhole routing, a worm that arrives at a node is immediately directed to one of the outbound links leaving that node, unless this node is its destination. Thus each worm, once injected into the network is never blocked or stored at an intermediate node. Rather, it keeps moving until it finds its destination. This principle of "ever-moving" is called the ***hot potato principle*** or the ***hot potato paradigm.*** Indeed, many parallel machines use variants of wormhole and hot potato routing. Note, however, that the hot potato principle may disturb the routing when a desired outbound link is already taken by another worm.

In this case, a worm is deflected to a distant region in the network which may not be on the way to its destination. Moreover, worms might deflect each other in an infinite cycle, causing livelock. The authors have proposed and analyzed routing algorithms based on this paradigm for $n \times n$ mesh and hypercubes.

4.7 Comparison of Switching Techniques

Figure 4.5 describes a hierarchy of switching techniques. It clearly shows the differences and similarities between various techniques. It can be seen that circuit-switching is basically a modified form of store-and-forward, in which the path is reserved prior to the transmission of data. Virtual cut-through is essentially a hybrid of circuit-switching and store-and-forward. If the packets encounter busy channels at all intermediate nodes, the outcome is exactly the same as store-and-forward. On the other hand, if all of the intermediate channels are free, the outcome is similar to circuit-switching. Wormhole switching is a modification of virtual cut-through with the added feature that packets are further broken into smaller units (flits) and provision is made to block the flits in-place, i.e., the flits are buffered in flit buffers of the node which they happen to traverse at that time. Moreover, the channels in wormhole switching cannot be used by any other message if they are being held.

In general store-and-forward is simple but it has two disadvantages:

(1) Each node must buffer every incoming packet, consuming memory space.

(2) The network latency is proportional to the distance between the source and destination nodes. The network latency is $(L/B)D$, where L is the packet

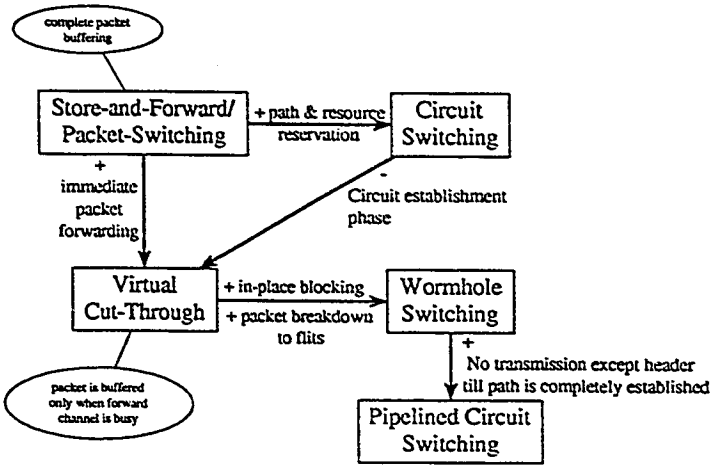


Figure 4.5: A hierarchy of switching techniques.

length, B is the channel bandwidth, and D is the length of the path between the source and destination nodes [107].

Circuit-switching is suitable for long messages and when the network load is low. Also, in circuit-switching deadlock-freedom is guaranteed.

One major advantage of virtual cut-through is that deadlock prevention is relatively easy because a blocked message does not tie up any channels, only buffers. A major disadvantage is that routers must be able to buffer a number of entire messages, increasing router size and perhaps reducing router speed [19].

Since virtual cut-through does not have to buffer the entire packet before forwarding it to the next node, the data-link level protocol is eliminated which is good for direct networks, as the network complexity and overhead are further reduced. However, because of the high transmission error rate, eliminating the data-link level protocol will delay the detection of a transmission error. The error will be detected by an end-to-end acknowledgment provided at the transport layer. This is the main

reason that virtual cut-through is not normally used [107]. An analytical model for the performance of virtual cut-through in hypercube and other interconnection networks is presented in [63].

Compared to the store-and-forward switching method that was used in early multicomputers, wormhole routing often reduces the effect of path length on communication latency. In the absence of channel contention, which occurs when two messages simultaneously require the same channel, the measured latency of a wormhole routed message has been shown to be nearly independent of the distance between the source and destination nodes.

In contrast with virtual cut-through, a wormhole router stops a message "in-place" when its head is blocked. The head of the message stops, and the remainder of the message is stopped, holding the buffers and channels along the path it has already formed. The primary advantage of wormhole routers is that router buffer requirements can be small, allowing routers to be extremely small and fast. The obvious disadvantage is that by allowing a message to retain the buffers and channels, wormhole routing increases the possibility of resource cycles which cause deadlock [19]. Wormhole routing is attractive in that:

(1) it reduces the latency of message delivery compared to store-and-forward routing, and

(2) it requires only a few flit buffers per node.

Virtual cut-through differs from wormhole routing in that it buffers messages when they block, removing them from the network. The deadlock properties of

cut-through routing are accordingly identical to those of store-and-forward. With wormhole routing, blocked messaged remain in network (i.e., in flit buffers).

In general, store-and-forward routing is a simple technique which works well when the packets are small in comparison with the channel widths. Both wormhole routing and virtual cut-through routing perform well with long messages, assuming they encounter little or no congestion. Under heavy traffic conditions, however, virtual cut-through performs significantly better than wormhole routing due to the fact that blocked packets are stored in only one node. The primary disadvantage of virtual cut-through is that it requires significantly more storage than wormhole routing [25]. A summary of the above comparison is presented in Figure 4.6.

4.8 Virtual Channels

Virtual channels were first proposed by Dally & Seitz [30] to allow the design of deadlock-free routing algorithms. As virtual channels provide an inexpensive mechanism to increase the number of logical channels without adding more wires, several adaptive routing algorithms based on them have been proposed.

Adding virtual channels to an interconnection network is analogous to adding lanes to a street network. A network without virtual channels is composed of one-lane streets. In such a network, a single blocked packet blocks all following packets. Adding virtual channels to the network adds lanes to the streets allowing blocked packets to be passed. In addition to increasing throughput, virtual channels provide an additional degree of freedom in allocating resources to packets in the network. This flexibility permits the use of scheduling strategies, such as routing the oldest

Switching Technique	Advantages	Disadvantages
Store-and-Forward	(1) No need for repeated attempts to establish circuit. (2) Simple (3) Tight upper limit on packet size makes sure that the transmission line cannot be monopolized by one processor. (4) Well-suited to interactive traffic. (5) Does not statically reserve bandwidth, rather acquires and releases it as necessary, which may be utilized by other packets from unrelated sources goin to unrelated destinations.	(1) Needs buffer for every packet. (2) Network latency is proportional to distance between source and destination. (3) Increase in load results in congestion due to buffers being exhausted. (4) Potential deadlock.
Circuit Switching	(1) Suitable for long messages. (2) Deadlock-free	(1) Statically reserves the required bandwidth in advance, so any unused bandwidth on an unallocated circuit is just wasted. (2) Waste of bandwidth in case of circuits being completely relinquished when unable to fully establish.
Virtual Cut-Through	(1) Good for long messages. (2) Delay due to unnecessary buffering in front of an idle channel is avoided. (3) Deadlock prevention is relatively easy. (4) Elimination of data-link protocol. (5)Performs better than wormhole routing under heavy load.	(1) Routers must be able to buffer a number of entire messages when the channels are busy. (2) It buffers messages when they block, removing them from the network, wasting bandwidth. (3) Restricted to profitable routing only.
Wormhole Routing	(1) Reduces the effect of path length on communication latency. (2) Router buffer requirements are low, allowing routers to be small and fast. (3) Good for long messages.	(1) Increased possibility of resource cycles causing deadlock. (2) Inability to support backtracking.
Pipelined Circuit-Switching	(1) Supports backtracking. (2) Supports misrouting. (3) Deadlock free. (4) Realizes maximum reliability.	(1) Large setup time (2) Increased number of virtual channels. (3) Reserves physical channels for long duration thus affecting performance. (4) Increased header size due to carrying history information.

Figure 4.6: A comparison of switching techniques.

packet first, that reduce the variance of network latency [28].

Virtual channels were first implemented in the torus routing chip [31]to provide deadlock-free routing. The J-machine network [28] uses virtual channels to provide two logical networks on a single physical network. The iWarp processing element uses virtual channels to guarantee bandwidth to virtual circuits.

4.8.1 The Concept of Virtual Channels

A source buffer, a destination buffer and some channel state make up a ***virtual channel.*** It is a channel from the point of view of resource allocation, but may share a physical communication channel with other virtual channels. Virtual channels, because they deal with routing, are allocated packet-by-packet. The buffers associated with a virtual channel remain allocated to a packet for its duration. They are allocated to the individual flits of the packet on a flit-by-flit basis. The physical channel is also allocated flit-by-flit. When the leading or head flit of a packet arrives at a channel, it requests a virtual channel. Once the channel is allocated, it will remember the next step of the route (i.e., which outgoing channel to use) until the tail flit of the packet is encountered. Starting with the head flit, each flit in turn must compete for buffer and channel resources. For a flit to advance, three conditions must hold: (1) It must be resident in a source buffer; (2) it must be allocated an empty destination buffer; and (3) it must be allocated use of the physical channel [26].

Each physical link is divided into some fixed number of unidirectional ***virtual channels.*** Each virtual channel has its own flit buffer, control, and data path. In

some designs, such as those of the Intel Touchstone and Intel/CMU iWarp, several unidirectional channels in the same direction share a single physical unidirectional channel; in other designs, unidirectional virtual channels in opposite directions share a physical bidirectional channel [107]. Each channel can carry data for one virtual circuit, i.e., one path. Thus, a circuit from one node to another consists of a sequence of channels on the links in the path between the nodes. The circuit from A, B C and D in Figure 4.7 all use the physical link $X - Y$ to communicate across. When node X sends data to Y, the latter must determine the circuit associated with the data. Two commonly used techniques for providing this information are:

(1) Divide the link into a fixed number of time slots and statically assign each time slot to a channel. The time slot on which the data arrive identifies the sending channel. This is called ***static time-division multiplexing.***

(2) Precede the data with a tag that identifies the channel on which it is being sent. In this scheme, the available bandwidth on the link is allocated to the various channels by some demand-driven scheduling algorithm. This is called ***demand-driven time-division multiplexing.***

In the first scheme, the link is effectively divided into several lower bandwidth links, with the sum of these bandwidths equal to that of the physical link. This approach is sometimes used in circuit-switched transport mechanism. If a channel does not send any data, its allocated bandwidth is wasted. In addition, latency is increased because each channel must wait for its turn to send a unit of data. In the second scheme, the entire bandwidth of the link can be allocated to channels

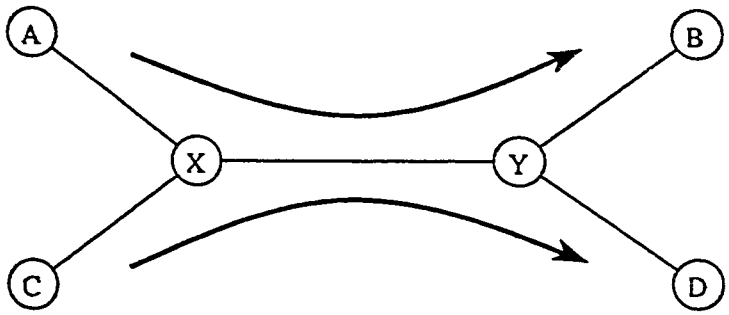


Figure 4.7: Two paths multiplexed through the same link.

when they have data to send, so the inefficiencies associated with the previous approach are avoided. However, some bandwidth is required to carry the channel tag. Demand-driven time-multiplexing is superior if the degree of multiplexing on each link is high and communication is bursty. This is often the case in multicomputer communications, so the demand-driven approach is more suitable [115].

Through virtual channels, a physical network can be divided into multiple disjoint networks, thereby facilitating adaptive routing algorithms. Virtual channels are useful in three other ways. First, by increasing the degree of connectivity in the network, they facilitate the mapping onto a particular physical topology of applications in which processes communicate according to another logical topology. For example, an application in which processes communicate according to a hexagonal array can be mapped onto a 2D mesh. Second, even when the application and the architecture have the same topology, extra connections may still be needed to route around congested or faulty nodes. Third, virtual channels provide the ability to deliver guaranteed communication bandwidth to certain classes of packets. For example, it is important that some bandwidth be reserved to support system-related functions, such as debugging, monitoring, and system diagnosis. By time-multiplexing virtual

channels onto physical channels using a fair schedule, availability of some minimum bandwidth can be guaranteed to each virtual channel as long as the number of virtual channels sharing the same physical channel is bounded.

The most important issue concerning virtual channels is the multiplexing and arbitration of a physical channel among many virtual channels. The multiplexing technique should be designed to maximize channel utilization. Specifically, if m virtual channels share a physical channel with bandwidth W, and k virtual channels are active, where $1 \leq k \leq m$, then each active virtual channel should have an effective bandwidth of W/k. Since the number of active virtual channels is a function of time, the router should be able to dynamically allocate channel bandwidth to the active virtual channels.

An example of the use of virtual channel is depicted in Figure 4.8. In the figure, a fragment of a network is depicted with a rounded box denoting a node, a solid arrow denoting a channel between two nodes, and a box denoting a flit buffer. Dashed arrows denote routes that are in progress. East, west, north, and south (E, W, N, S respectively) represent the usual directions in two-dimensions. Packet A is blocked holding buffer 3E (east side of node 3) and 4S. Packet B is unable to make progress even though all physical channels it requires, 1E to 2W through 4E to 5W, are idle because packet A holds buffer 3E which is coupled to channels 3E to 4W. Figure 4.8(b) illustrates the addition of virtual channels to the network of Figure 4.8(a). Packet A remains blocked holding buffers 3E.1 and 4S.1. However, packet B is able to make progress because buffer 3E.2 is available allowing it access to channel 3E to 4W [28].

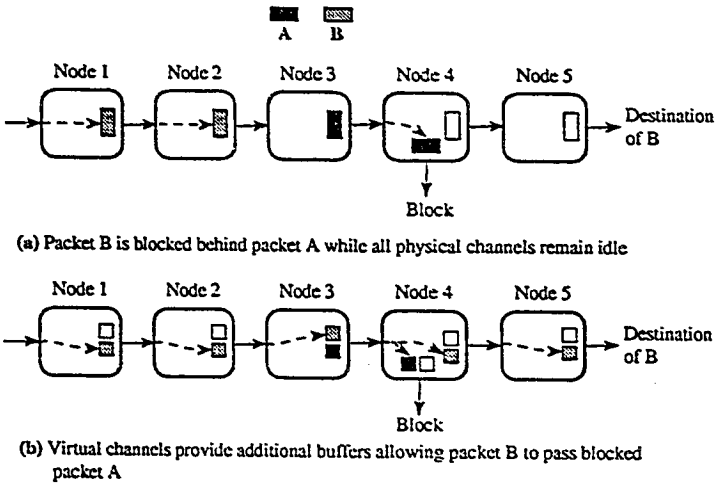


Figure 4.8: Removing deadlock using virtual channels.

4.8.2 Virtual Channel and Deadlock-Avoidance

With virtual channels, deadlock is avoided by making the routing relation acyclic. The resources being allocated are virtual channels which are flit buffers at either end of a physical communication channel and some associated routing state. If several virtual channels are associated with a single physical channel, several packets can use the same physical communication resource without interacting from the point of view of deadlock avoidance.

Each node of an interconnection network contains a set of buffers and a switch. The buffers are usually partitioned into sets associated with each input channel, an input-buffered node, as shown in Figures 4.9 and 4.10.

An output-buffered switch can be considered to be an input-buffered switch with a non-blocking first stage by associating the buffers on the output of each stage with the inputs of the next stage. A conventional network organizes the flit buffers associated with each channel into a first-in, first-out (FIFO) queue as shown

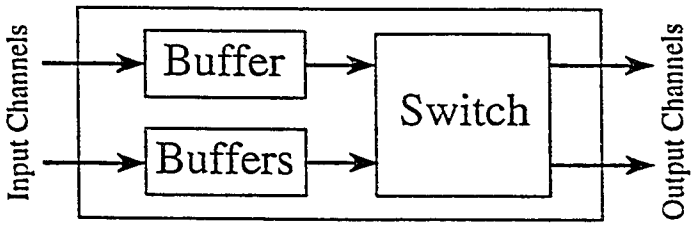


Figure 4.9: Node organization. Each network node contains a set of buffers for each input channel and a switch.

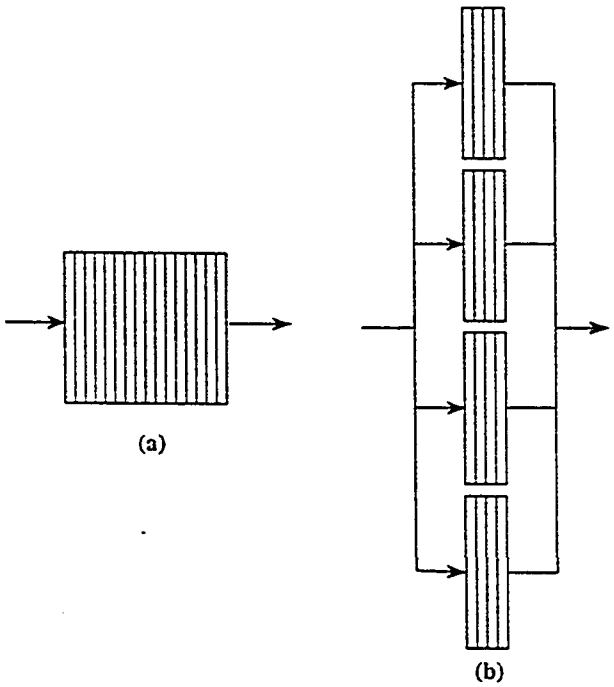


Figure 4.10: (a) Conventional nodes organize their buffers into FIFO queues restricting routing. (b) A network, using virtual-channel flow control, organizes its buffers into several independent lanes.

in Figure 4.10(a). This organization restricts allocation so that each flit buffer can contain only flits from a single packet. If this packet becomes blocked, the physical channel is idled because no other packet is able to acquire the buffer resources needed to access the channel.

A network using virtual channel flow control organizes the flit buffers associated with each channel into several lanes as shown in Figure 4.10(b). The buffers in each lane can be allocated independently of the buffers in any other lane. This added allocation flexibility increases channel utilization and thus throughput. A blocked message, even one that extends through several nodes, holds only a single lane idle and can be passed using any of the remaining lanes.

Another view of this application of virtual channel to deadlock avoidance in 4-cycle is shown in Figure 4.11(a). The 4-cycle is shown as a traffic circle with four entrances. The flit buffers correspond to the areas of roadway between the entrances. To indicate that wormhole routing is used, packets are shown as worms. When two packets enter the circle, each is allocated two flit buffers. "Wormlock" then occurs because the next packet buffer requested by each worm is occupied by the other. Adding two virtual channels Figure 4.11(b) converts the traffic circle into a spiral. Because there is no dependency cycle, one packet is able to progress, but the other packet remains blocked [26].

4.8.3 Performance of Virtual Channels

The development of virtual channels made possible the use of wormhole routing in cyclic networks with $k > 2$. Before the invention of virtual channels, it was not

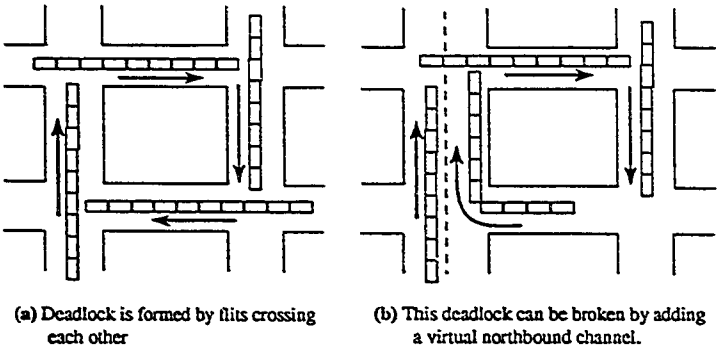


Figure 4.11: Removing deadlock with virtual channels.

understood how to avoid deadlock in such networks. The application of virtual channels extends beyond deadlock avoidance, however. They are a versatile mechanism that can be applied to solve a number of network problems. They can be useful in increasing network throughput by reducing physical-link idle time. Only a few flits per node of buffering are required. Because of this small memory requirement, no node memory space or bandwidth is needed for through messages, and the network can operate at a speed independent of the node memory [26].

Virtual channels drastically reduce channel contention by offering a higher number of free channels to route a message towards its destination. Then, header delay is reduced. However, when several message headers are routed through virtual channels belonging to the same physical channel, bandwidth is shared among those messages. There is another benefit of using virtual channels. When a message header is blocked, the channel bandwidth used by that message along its path is freed. It can be used by other messages sharing the same physical channels, provided they are not blocked [28].

Virtual channels can be used to duplicate a network logically by doubling the

number of flit buffers. The resulting two networks are logically separate, i.e., traffic in one will not block traffic in the other. They can be used to implement two priorities of message delivery. In addition to duplicating the network, one may add services to one of the logical networks sharing physical channels. For example, the original proposal for IBM RP3 called for two networks, a normal network and a combining network, to handle accesses to synchronization variables. Rather than duplicating the costly physical channels, these two networks could be implemented with virtual channels sharing the same set of physical channel [26].

The virtual channel concept is not without drawbacks. As the number of virtual channels increases, the scheduling becomes more complicated, requiring additional hardware complexity and potentially increasing network latency. The sharing of bandwidth may also increase latency. Consider the following scenario in which a communication path traverses multiple physical channels, each of which supports many virtual channels. If the bandwidth of each physical channel is W and there is no sharing with other virtual channels, the effective bandwidth of the communication path is W. On the other hand, if one of the physical channels along the path is shared with three other packets, that channel becomes a bottleneck and the effective bandwidth of the entire path is reduced to $W/4$, even though the available bandwidth of all other channels in the path is W. The trade-off between increased network throughput and longer communication latency should be considered when deciding whether to use virtual channels [107].

Duato [39] has analyzed the performance of virtual channels in detail. It has been suggested that the larger the number of virtual channels, the higher the performance.

This is however not true for all the topologies and network traffic conditions. A first approach consists of choosing the optimal number of virtual channels for a given network. However, that optimal number depends on several parameters, including network traffic. Then, for networks with a varying amount of traffic, the optimal number of virtual channels cannot be properly tuned.

A very interesting characteristic of virtual channels is that benefits become more noticeable for medium to high network traffic. The drawbacks mainly appear for low network traffic. When the traffic is low, the probability of finding a busy channel is very low, but not null. When a message header requests a busy virtual channel, it may use another virtual channel belonging to the same physical channel. Channel contention has been eliminated but channel multiplexing will slow down both the messages, even if they do not share any other channel with other messages.

4.8.4 Extension of Virtual Channel Concept to Virtual Networks

In order to free the communication system from the constraints of a fixed physical topology, Linder & Harden [94] introduced the concept of ***virtual networks*** which extends the notion of virtual channel to multiple, virtual communication systems that provide adaptability and fault-tolerance in addition to being deadlock-free.

Figure 4.12 illustrates the concept of virtual communication systems underlying a single physical communication system.There is a mapping from the virtual nodes and channels to the physical nodes and channels although it is not necessarily one-to-one. The only restriction on the mapping is that nodes adjacent in the virtual

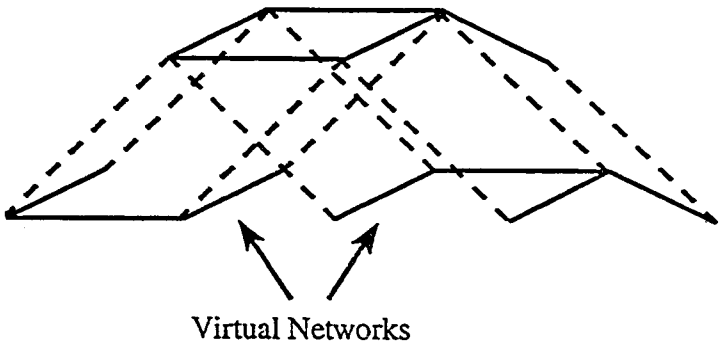


Figure 4.12: Multiple virtual networks underlying a single physical network.

systems map to nodes adjacent in the physical system. All virtual channels that map to a single physical channel are time multiplexed over the physical channel. Some relevant definitions are as follows:

> ***Physical Interconnection Network***: A ***physical interconnection network***, PI, is a strongly connected directed graph, $PI = PG(PN, PC)$. PN is a set of nodes representing processors, and PC is a set of edges representing actual physical channels connecting the processors.

Let pn_i and pc_i be the ith node and channel, respectively. Also, the source node of pc_i is ps_i and the destination node is pd_i.

> ***Virtual Interconnection Network***: A ***virtual interconnection network***, VI_i, is a strongly connected directed graph, $VI_i = VG_i(VN_i, VC_i)$. The subscript i is used because a single physical interconnection network could have several virtual interconnection networks mapped to it. VN_i is a set of virtual nodes that are mapped to PN by the mapping function $nmap : VN_i \rightarrow PN$. VC_i is a set of edges representing virtual channels that are mapped to PC by the function $cmap : VC_i \rightarrow PC$.

Let vc_{ij} be the jth virtual channel in the ith virtual interconnection network. Associated with each virtual channel, vc_{ij}, is a queue vq_{ij}. Also, the source node of vc_{ij} is vs_{ij} and the destination node is vd_{ij}.

Connection: A ***connection*** exists from channel vc_{ij} to vc_{ik} if the flit at the head of vq_{ij} will be transferred to the tail of vq_{ik} when the space in vq_{ik} become available.

Channel Dependency Graph: A ***channel dependency graph***, D_i, for the virtual interconnection network VI_i is a graph $D_i = G_i(VC_i, E_i)$ where $E_i = \{(vc_{ij}, vc_{ik}) \mid \text{a connection can exist from } vc_{ij} \text{ to } vc_{ik}\}$. Thus, D_i is a graph with a node for every channel in VI_i and edges to define what connections can be made between the channels. A channel is not allowed to connect to itself, so there will be no 1-cycles in Di.

Routing Function: A ***routing function*** $R_i : VC_{ij} \times \{\textit{all possible states of } VI_i\} \rightarrow VC_{ik}$ for a virtual interconnection network VI_i maps a channel vc_{ij} and the present state of VI_i to another channel vc_{ik}. When the first flits of a message arrive at the head of some channel queue, vq_{ij}, the routing function determines the channel, vc_{ik}, that vc_{ij} should be connected to so that the flits in vq_{ij} can advance toward their destination.

The approach used in this scheme highlights a separation between the structure of the virtual interconnection networks and the adaptive routing algorithms that indicate how messages will travel. Thus, instead of showing each new routing function is deadlock-free, the virtual interconnection network is shown to be deadlock-free

for a given channel dependency graph, and then new routing functions are shown to obey the restrictions of the channel dependency graph.

To illustrate the concept, an analogy can be drawn between the virtual interconnection networks and traffic systems. The virtual interconnection network is a map that tells what channels (roads) exist, and the channel dependency graph tells what connections (turns) are legal. Then if all messages (cars) obey these restrictions and the restrictions were properly developed, no deadlock (accidents) should occur. A routing function (the driver) just guides a message from node A to node B while obeying all restrictions.

4.9 Summary

In this chapter we have discussed the various switching techniques used in multicomputer networks. We also compared the switching techniques in the light of their advantages and disadvantages and the differences between them in terms of path establishment, resource management and buffer allocation policies. We have also shown the interrelationship among the different switching mechanisms with respect to their similarities and how one can be derived from the other. Also described in detail is the concept of virtual channels, how they are used in providing deadlock-freedom and how the concept is extended to develop the idea of virtual networks. A discussion of the performance of virtual channels has also presented.

Chapter 5

Routing Techniques for One-to-One Communication

In this chapter the routing techniques for one-to-one communication are studied. They are organized according to the classification proposed in Chapter 3. This chapter is organized into five sections as follows: oblivious routing, completely adaptive minimal routing, completely adaptive non-minimal routing, partially adaptive, minimal routing, partially adaptive, non-minimal routing.

5.1 Oblivious Routing

This section consists of the description of three techniques: e-cube routing, virtual channel based e-cube routing and dimension-ordered routing.

E-cube Routing

The *e-cube* routing scheme [82] is an oblivious routing technique proposed for

binary hypercubes as a deadlock-free routing technique. It examines at most a single path, selected by *a priori* knowledge of the hypercube topology. In the binary hypercube [119] topology, the labels for adjacent nodes differ by a single digit in their binary representation. The transmission link between two adjacent nodes is labeled with the position of the differing bit. If nodes are k hops apart, they differ in k bit positions. The links corresponding to those bit positions are called ***candidate links***, and are the only links that will reduce the distance to the destination node. At each routing step, the e-cube routing scheme considers a single candidate link. If that candidate link is in use, the circuit is blocked. The existing circuit is torn down and message transmission is reattempted.

Dimension-Ordered Routing

A channel numbering scheme often used in n-dimensional meshes is based on the dimension of channels. In ***dimension-ordered routing*** [82], each packet is routed in one dimension at a time, arriving at the proper coordinate in each dimension before proceeding to the next dimension. By enforcing a strict monotonic order on the dimensions traversed, deadlock-free routing is guaranteed. Since each node in a 2D mesh is represented by its position (x, y), the routing is called xy-routing. In this scheme, the packets are first sent along the x dimension and then along y dimension.

Virtual Channel E-cube Routing

Dally & Seitz [30] presented a general method for constructing deadlock-free routing algorithms for arbitrary wormhole-routed networks. This method is based on the concept of virtual channels. Designing of further deadlock-free routing algorithms for wormhole routing was simplified further by the authors with a proof that

an acyclic channel dependency graph guarantees deadlock-freedom. Each node of the channel dependency graph is a virtual channel. There is a directed edge from one virtual channel to another if a message is permitted to use the second virtual channel immediately after the first. Since the graph is acyclic, deadlock-freedom can be shown by assigning a numbering to the edges of the graph, ensuring that virtual channels are always used in increasing or decreasing order. The authors proposed their proof technique for deterministic routing algorithms, which are characterized by functions of the form, $R : C \times N \times N \rightarrow C$, where the input channels C, and the current and destination nodes, belonging to the set of nodes N, define an output channel on which to route the message. An acyclic dependency graph has also been used as a basis for developing adaptive routing algorithms defined by relations of the form $R : C \times N \times N \rightarrow C^p$, where a set of output channels, rather than a simple channel, is defined on which to route a message.

A deadlock-free routing algorithm is also presented for k-ary n-cubes which is an extension of the e-cube routing algorithm using the technique of virtual channels. Each node of a k-ary n-cube is identified by an n-digit radix k number. The ith digit of the number represents the node's position in the ith dimension. Channels are identified by the number of their source node and their dimension. To break cycles, each channel is divided into an upper and lower virtual channel. To assure that the routing is deadlock-free, it is restricted to route through channels in order of descending subscripts. As in the e-cube algorithm, routing is performed in order of dimension, most significant dimension first. In each dimension i, a message is routed in that dimension until it reaches a node whose subscript matches the destination

address in the ith position. The message is routed on the high channel if the ith digit of the destination's address is greater than the ith digit of the present node's address. Otherwise the message is routed on the low channel. It is easy to see that this routing algorithm routes in order of descending subscripts, and is thus deadlock-free.

5.2 Completely Adaptive, Minimal Routing

This section consists of the description of the following techniques: NRCC and related techniques, idle routing, mad postman routing, Duato's protocol, three protocols routing, message flow model, hop based routing, hanging algorithm, adaptive set routing, opt-y routing, and exhaustive profitable backtracking.

5.2.1 Progressive Techniques

NRCC, Shortest Queue, Delta and Hybrid Weighted Routing

Kim & Reed [67] present four algorithms for adaptive packet-switched routing in hypercubes. These include the NRCC (network routing control center), shortest queue, delta and hybrid weighted routing. In NRCC routing, routing decisions are made at a central node, the Network Routing Control Center. Nodes report locally perceived network state information to the NRCC when it is necessary to update routing tables. With the reported global information, the NRCC generates new routing tables and sends the routing tables to local nodes. NRCC routing does not adapt quickly to instantaneous traffic changes.

Shortest queue routing sends packets on those links with the shortest queue lengths. Because a node does not know the global network state, packets are not guaranteed to monotonically decrease the distance to their destination. To avoid this, shortest queue routing may restrict the candidate links, based on the destination of the packet.

Delta routing exploits both global information and local information. A designated central node (NRCC) collects global information and selects paths based on global information. Contrary to NRCC routing which selects only shortest paths, delta routing selects multiple, acceptable candidate paths and delegates the final routing decisions to local nodes. A parameter δ, coordinates the relative routing responsibility of the NRCC and local nodes. The selected path set includes the shortest paths and all other paths whose costs differ from the cost of the shortest paths by at most δ. If only one path is selected for a certain destination then all packets destined for that node will be routed along the path. However, if more than one potential path is identified, a local node will select a path from the candidate paths based on the instantaneous local information.

To avoid choosing paths at local nodes which move the message away from its destination, ***hybrid weighted*** routing uses both global and local information in the local path selection procedure. Local nodes use both the global path length information and local queue length information to select paths.

Idle Routing

Grunwald & Reed [55] proposed a completely adaptive, profitable, progressive scheme called *Idle*, designed for circuit-switching networks. *Idle* traverses prof-

itable links in a fixed order, as does e-cube, until it encounters a busy link. In such a case, ***Idle*** examines the next profitable link in its order and takes that one if possible. ***Idle*** will continue to progress toward its destination as long as profitable links are available. If all profitable links are unavailable, it must block or tear down the path and retry. ***Idle*** requires no extra status bits in its routing header. If the scheme requires probes to block indefinitely, ***Idle*** can deadlock. The method used to prevent deadlock involves simply aborting and retrying whenever it could not route. The number of possible paths provided by the scheme is $n!$.

Mad Postman Routing

Jesshope, Miller & Yantchev [61, 101, 136] proposed a fully adaptive minimal routing algorithm for 2D meshes and toroidal networks. Their algorithm requires two virtual channels for each physical channel and the corresponding network is called a ***double-xy network***. The algorithm is called the ***mad postman*** algorithm which provides deadlock-freedom and low latency and is based on the concept of virtual networks. Deadlock is avoided without imposing restrictions on routing by splitting the virtual network into a number of acyclic virtual networks, either multiplexed onto the same channel resources or implemented directly in hardware. This allows messages to adapt to local traffic conditions, thus increasing the effective bandwidth of the network.

All packets can be divided into four classes according to the directions they need to be routed. These four classes correspond to the four quadrants of a 2D plane: $(+X,+Y)$, $(-X,+Y)$, $(-X,-Y)$ and $(+X,-Y)$. Four independent virtual networks, each routing messages in one of the four quadrants, can provide for the

necessary routing path.

The mad postman routing strategy reduces latency to an absolute minimum by propagating messages in an eager, send-first-decide-later fashion. It delivers the head of an unblocked message to its destination in a time independent of the channel width; only the throughput of messages is affected by the total network bandwidth. For minimum latency, the message must be routed along the dimension of the leading address digit. Latency will be further reduced, if when each packet has been fully routed in a particular dimension, then the associated digit is stripped off. This will also increase the efficiency of transmitting packets. If a packet cannot be routed in the first dimension, then it can either be routed in some other dimension or wait till the necessary channel is available.

Duato's Protocol

Duato [38] states a general theorem defining a criterion for deadlock freedom and then uses the theorem to propose a fully adaptive, profitable, progressive technique. Called ***Duato's Protocol*** (DP), the theorem states that by separating virtual channels on a link into restricted and unrestricted partitions, fully adaptive routing can be performed and yet be deadlock-free.

In essence, Duato's theorem allows the creation of hybrid schemes that operate over these distinct channel partitions. Duato proposed this in the context of binary hypercubes, which are easily extended to k-ary n-cubes. DP uses two virtual channels per physical channel for an e-cube routing subfunction (a routing function over a subset of channels) and the remaining virtual channels for an ***Idle*** [55] routing subfunction. The combined scheme attempts to route using ***Idle*** first. If it cannot

route using ***Idle***, e-cube routing is performed. If that fails, the scheme blocks. Duato's theorem guarantees that this scheme is deadlock-free. DP requires no extra status bits in its routing header.

Three Protocols Routing

A fully adaptive, deadlock-free minimal routing algorithm was presented by Su & Shin [126], that uses only one extra virtual channel as compared to the deterministic routing. The algorithm is based on wormhole routing.

According to the property of wormhole routing, a packet contains one or more flits. The flit has to build the path between the source and destination. Each flit of a packet following the header flit advances as soon as the proceeding flit moves and gets blocked when the required channel resources are unavailable. A simple examination of the protocol for requesting channel resources in multicomputers leads to two cases:

(1) **Request-then-wait:** While the requested channels are held by the other packets, the requesting packet will block and wait for the channels to be available.

(2) **Request-then hold:** While the requested channels are available, the packets will hold the channel and route the flits forward.

A third protocol is proposed as follows:

Request-then-relinquish: Once a packet fails to get its requested channels, it terminates the request and does not wait for these channels either. In other words, no packet waits for a channel using this protocol. As described

below, this protocol is only used to provide adaptive routing. In case no channel is available, the packet follows the other two protocols and undergoes a dimension-ordered routing.

Based on the above three protocols, an adaptive deadlock-free routing algorithm has been proposed with the following assumptions:

(1) The interconnection network can be divided into two virtual interconnection networks VIN_1 and VIN_2, where VIN_1 supports deadlock-free minimal routing and VIN_2 uses one virtual channel (called ***extra virtual channel***) to share the bandwidth of a physical link with the channels in VIN_1. The channels of VIN_2 are used to enhance routing and adaptability.

(2) Each virtual channel in VIN_1 is assigned a channel number. Also, packets requesting for the virtual channels in VIN_1 should obey strict increasing or decreasing order of channel number.

(3) Only request-then-hold and request-then-relinquish protocols are used on the extra virtual channel (VIN_2), i.e., a requesting packet never waits for the extra virtual channel. Also, only request-then-wait and request-then hold protocols are used on the virtual channels in VIN_1.

The algorithm is called ***three protocols routing*** or ***3P routing*** because it is based on the above three protocols. It is applied to nD meshes and it is applicable to k-ary n-cubes, and other topologies.

Message Flow Model

Lin, McKinley & Ni [92, 93] proposed an approach to deadlock-free routing in wormhole-routed networks called the ***message flow model.*** They also develop routing algorithms for 2D meshes and hypercubes based on the message flow model. The routing function for this model is defined as a relation $R : C \times V \times C$, where C is the set of channels and V is the set of nodes. The triple $(c_I, v_d, c_j) \in R$ implies that when the header flit of a packet destined for node v_d has arrived at a router from channel c_I, then it may be forwarded along channel c_j. Given a routing function R, a channel c_i and destination node v_d, the ***forwarding set***, $F_R(c_I, v_d)$ is defined as the subset of channels, $\{c_j | (c_I, v_d, c_j) \in R\}$, i.e., when a packet destined for node v_d has arrived on channel c_I, it may potentially be forwarded along any channel in $F_R(c_i, v_d)$ according to the routing function R.

Unlike the channel dependency graph model, which represents the *potential* routing of messages from one channel to another, the message flow model represents the *actual* flow of messages among channels by accounting for the manner in which a blocked message waits for a channel to become available.

The message flow model is based on the deadlock-freedom of the above routing function. To guarantee the deadlock-freedom of the routing function, the ***deadlock-immunity*** of a channel for a particular message is defined as: given a routing function R, a channel c is ***deadlock-immune*** for a message M if and only if, once M occupies the channel c, M will eventually leave channel c and release it. Furthermore, if a message M can never use channel c, c is also defined to be deadlock-immune for message M. A channel is said to be deadlock-immune if and only if it

is deadlock-immune for all messages when using the routing function R.

Let M be a message arriving on channel c_I and destined for node v_d, and let R be the routing function. The ***waiting channel***, denoted $f_R(c_i, v_d)$, where $f_R(c_i, v_d) \in F_R(c_i, v_d)$ and $F_R(c_i, v_d) \neq \emptyset$, is defined as the channel on which M will wait if all of the channels in $F_R(c_i, v_d)$ are busy.

The crux of the message flow model is the following theorem which shows how to identify a deadlock-immune channel given other known deadlock-immune channels, as well as how to use forwarding sets to identify deadlock-immune channel for a given routing function.

> Given a routing function R, a channel c_i is deadlock-immune if and only if, either
>
> (a) for every node $v_d \in V$, the forwarding set $f_R(c_i, v_d) = \emptyset$, or
>
> (b) for every $v_d \in V$ with $F_R(c_i, v_d) \neq \emptyset$, and every channel $c_j = (v_j, v_{j+1})$, $v_{j+1} \neq v_d$ such that c_j is in path p, $p \in P_R(c_i, v_d)$ (denotes the set of all paths from channel c_I to node v_d permitted under R) channel $f_R(c_i, v_d)$ is deadlock-immune.

Based on the above model, an enhanced version of the double y-channel algorithm (i.e., multiple virtual network scheme) of Linder & Harden [94] is presented for 2D meshes, in which a network is partitioned into two subnetworks, namely, the east subnetwork and the west subnetwork; each pair of nodes neighboring in the Y dimension are connected by a pair of channels, while each pair of nodes neighboring in the X dimension are connected by a single channel. Any westbound message

will be routed through west subnetwork and any eastbound message will be routed through the east subnetwork. The double y-channel algorithm is a minimal fully adaptive routing algorithm. The message flow model increases the adaptivity of the double y-channel algorithm without adding more virtual channels. Specifically, a westbound message may also safely use y-channels in the east subnetwork, if such a channel is available when the message arrives at the corresponding router. However, when a westbound message is blocked, it must always wait for a channel in west subnetwork.

A fully adaptive deadlock-free algorithm is also developed for hypercubes. Each physical channel is assumed to have two virtual channels. One channel of each pair is called ***fixed*** and the other ***free***. When a message is blocked, it always waits for a fixed channel. A fixed channel is used in such a way that it is shown to be deadlock-immune; a free channel can be used without restriction. Thus if each fixed channel is deadlock-immune, then so is each free channel. The hypercube algorithm requires only two virtual channels regardless of the size of the hypercube, compared with the scheme of Linder & Harder [94], which requires 2^{n-1} virtual channels per physical channels.

Hop Based Routing

Boppana & Chalsani [8, 9] present a framework to design fully-adaptive, deadlock-free wormhole routing algorithms for a variety of network topologies. A number of wormhole routing algorithms based on store-and-forward hop-based schemes have been designed. These algorithms are called ***hop schemes*** because they are based on the number of hops taken by messages. In this scheme, the network is partitioned

into several subsets, such that no subset contains two adjacent nodes. If C is the number of subsets, then the subsets are labeled $0, 1, \ldots, C-1$, and the nodes in subset i are labeled i. A hop is a ***negative hop*** if it is from a node with a higher label to a node with a lower label; otherwise it is a ***nonnegative hop***. One of these schemes is the ***positive hop scheme***, which provides $n\lfloor k/2 \rfloor + 1$ virtual channels on each physical link of a k-ary n-cube. A message reserves virtual channel i at an intermediate node to complete hop $i+1$. Another scheme is the ***negative hop scheme*** which provides $\lceil n\lfloor k/2 \rfloor / 2 \rceil + 1$ virtual channels on each physical channel. When a message is generated, the total number of negative hops taken is set to zero, and the current host is set to the source node. The message when it moves from an even node to an odd node, reserves a virtual channel of the same class it reserved in the previous hop; otherwise, it reserves a virtual channel one class higher that what it reserved in the previous hop.

Hanging Algorithm

Pifarré *et. al.* [112] proposed a number of routing algorithms for store-and-forward interconnection networks such as hypercubes and two-dimensional meshes. The techniques are applicable to virtual cut-through routed networks as well. They are all fully adaptive, deadlock- and livelock-free. The routing algorithm for hypercubes consists of a routing function that results from routing over the hypercube as ***hung*** from node $00\ldots0$. The main idea on which this ***hanging*** algorithm is based is the following. Each message is routed in two phases: In phase A, each message starts heading to node $11\ldots1$. So, in phase A, each message turns the incorrect 0's in the address of its source node into 1's.

In phase B, every message arrives at its destination by moving towards node $00\ldots0$. So, in this phase, each message turns the incorrect 1's of its source address into 0's. Therefore, all the required corrections are terminated at the end of this phase. Consequently, each message arrives at its destination.

An algorithm for two-dimensional meshes based on the idea of ***hanging*** the mesh from the (0, 0) and $(n-1, n-1)$ nodes consists also of two phases. In phase A the messages move towards their destination by visiting nodes in such a way that if a message passes from (x, y) to (x', y') in one routing step, then $x < x'$ or $y < y'$, i.e, the mesh is hung from node (0, 0) in phase A and the messages visit nodes with higher level, where the level of (x, y) is $x+y$. In phase B, messages visit nodes with lower number instead of those with higher number, i.e., the mesh is hung from node $(n-1, n-1)$ and the nodes are visited in decreasing level order. Each message starts the routing process in phase A. After all of the steps that could be taken in phase A have been completed, the message enters phase B. Certain messages arrive at their destination by taking steps only in phase A or phase B. The original scheme is partially adaptive which is extended to allow full adaptivity by allowing messages that have not finished their first phase to take phase B steps. Maximum length that any path may have under hanging algorithm is $O(log\ n)$ for hypercube and $O(n)$ for mesh, where n denotes the number of nodes in the network.

Adaptive Set Routing

Lin & Lin [89] proposed a minimal, fully adaptive wormhole routing algorithm for hypercubes which uses $(\lceil (n+1)/2 \rceil)$ virtual channels per physical channel where n is the number of dimensions. The routing algorithm is defined in two parts. The

first part is an ***adaptive set function*** that determines the set of virtual channels for the header flit to go one step further based on the current processor address, destination processor address and some routing information the header flit carries. The set of virtual channels determined thus is called the ***adaptive set.*** The second part is an ***output selection policy*** that selects a virtual channel from the adaptive set to route according to the network congestion information.

To avoid deadlock, and provide fully adaptive routing in an n-dimensional hypercube each physical channel is split into $(\lceil (n+1)/2 \rceil)$ virtual channels. These virtual channels form $(\lceil (n+1)/2 \rceil)$ virtual networks. Initially the message is routed in the 0th virtual network. During the routing, a message may transfer to the next virtual network only when the previous channel direction is negative (if (u, v) is a dimension k channel, i.e., $u \oplus v = e^k$ (e^k is a binary string whose bits are all 0 except the kth bit), then the direction of (u, v) is positive if the kth digit of $u = 0$ otherwise it is negative) and the message chooses positive direction to route in the next step. The message always gets closer to the destination in the routing steps.

The algorithm is extendible to k-ary n-cubes with number of virtual channels equal to $(\lceil (\lfloor k/2 \rfloor n+1)/2 \rceil)$. For general multicomputers, and fully adaptive wormhole routing, $(\lceil (d+1)/2 \rceil)$ virtual channels are needed per physical channel where d is the diameter of the network.

Opt-y

Schwiebert & Jayasimha [121] presented a deadlock-free fully adaptive minimal routing algorithm for meshes that is optimal in the number of virtual channels required and in the number of restrictions placed on the use of these virtual channels. It

is also proved that, ignoring symmetry, this routing algorithm is the ***only*** fully adaptive routing algorithm that achieves these goals. The technique requires $4n-2$ virtual channels for an n-dimensional mesh. In addition, if more than the minimum number of virtual channels is available, the routing algorithm can use these additional channels with the fewest possible number of restrictions. The scheme has also been generalized to n-dimensional meshes.

The technique is basically an extended version of the turn model [52]. This technique requires two virtual channels (numbered first and the second) in the North and South directions. The constraints imposed by the scheme is that a message which needs to route further in the West direction cannot use the first virtual channel in the North or the South direction.

5.2.2 Backtracking Techniques

There is only one completely adaptive, minimal backtracking technique presented in the literature, which is described as follows.

Exhaustive Profitable Backtracking (EPB), proposed by Grunwald & Reed [55], performs a straightforward depth-first search of the network using only profitable links. It is guaranteed to find a minimal path if one exists. With the history information distributed throughout the nodes, the probe consists of a vector of dimension offsets and a backtrack flag. This scheme is completely adaptive, profitable and backtracking. The number of possible paths provided by the scheme is $n!$.

5.3 Completely Adaptive, Non-Minimal Routing

This section consists of the description of the following techniques: chaos routing, one-pass deflection routing, dimension reversal routing, and exhaustive misrouting backtracking.

5.3.1 Progressive Techniques

Chaos Routing

Konstantinidou and Snyder [72, 73, 74, 75] advocate the use of a nonminimal, randomized, adaptive packet-switched routing technique for binary hypercubes. It is applicable to the more general k-ary n-cubes. There is no need for virtual channels in this technique.

In chaos routing, when a node becomes congested and the need for message misrouting arises, the message to be misrouted is randomly selected out of the node's full queue. The process effectively guarantees that the probability of an infinite message path is zero. The technique is deterministically deadlock-free and probabilistically livelock-free. It also prevents starvation from occurring. The chaos router implements virtual cut-through routing with fixed size packets.

The chaos router is an asynchronous router. Not only do nodes operate independently of each other, but even channels operate independently of each other. This asynchrony makes all of the routers effectively independent of one another. Moreover, the circuitry of the routers is selftimed, i.e., not clocked. Thus the routers

will run at different rates because of differences in the tasks being performed as well as variations in the electronics. A chaos router explicitly randomizes its message selection during derouting. The principal effect of this chaos, besides assuring network unpredictability and removing the overhead of livelock protection, is to make the router probabilistically livelock-free, meaning that the probability of a message remaining in the network longer than t seconds goes to zero as t increases. The intuition is that the router exploits randomness to keep the traffic chaotic, thereby causing the repetitious routing patterns, typical of livelocked messages, to decay.

One-Pass Deflection Routing

Greenberg & Hajek [54], drew some motivation from queuing theory where if there are as many resources as jobs then some jobs may not be granted their desired resources but rather be allocated some other resources temporarily. Such jobs are called ***deflected jobs***. Similarly, in ***deflection*** routing, nodes attempt to route each packet along a shortest path to its destination. However, when a node holds two or more packets whose desired paths call for the same link, the node grants the link to one of the packets and grants other links to the other packets. In this way, congestion causes packets admitted to the network to be misrouted temporarily.

Under the ***one-pass deflection*** routing, packets are assigned to the outgoing links in the following way. The packets are considered one at a time, in random order, with all orders being equally likely. Hence a given packet is considered, the node examines which outgoing links are on the shortest path to the packet's destination and sees if any of those are free, i.e., not already assigned to another packet. If at least one preferred link is free, the packet is assigned to a preferred, free link

at random, all choices being equally likely. Otherwise, the packet is assigned to a free but no preferred link, all choices being equally likely, and the packet is said to be deflected. The computational complexity of the technique is $O(d/2)$ and the maximum number of misroutes that a packet may suffer is $O(log\ d)$, where d is the number of packets transmitted by a node in unit time.

Dimension Reversal Routing

Dally & Aoki [29] proposed two deadlock-free adaptive routing algorithms for mesh networks which can be used with wormhole routing. Both algorithms permit misrouting, avoid deadlock using virtual channel to eliminate cyclic dependency, and introduce the use of ***dimension reversal (DR) numbers*** to break dependency cycles. The dimensional reversal number of a packet is the count of the number of times a packet has been routed from a channel in one dimension, p, to a channel in a lower dimension $q < p$. Dimension reversal numbers are assigned to packets as follows:

(1) All packets are initialized with a DR of 0.

(2) Each time a packet routes from a channel $c_I \in C'_p$ to a channel $c_j \in C'_q$, where $p > q$, the DR of the packet is incremented (where C'_p is the subset of virtual channels in dimension p and C'_q the subset of virtual channels in dimension q).

The first algorithm, called ***static***, labels each virtual channel on a physical channel with a DR number. Every time a message switches from a higher dimension to a lower dimension, the message must move to a virtual channel with a higher DR

number. Once a message has reached a virtual channel with the highest DR number, the message must use dimension-order routing.

Fully adaptive routing on a $k \times k$ mesh requires as many as k dimension reversals, so k virtual channels per physical channel are required for fully adaptive routing using the static algorithm. The static assignment of DR numbers to virtual channels restricts the number of DRs permitted and makes inefficient use of virtual channels. A packet may be blocked waiting for a virtual channel in its DR class, while other virtual channels for the same physical channel remain idle.

The ***dynamic*** algorithm overcomes these limitations by permitting packets to use any available virtual channel. Packets are labeled with their DR numbers and are not permitted to wait for a virtual channel held by a packet with a lower DR number. If all available virtual channels are occupied by packets with lower DR numbers, the packet reverts to dimension-order routing on an additional set of virtual channels reserved for this purpose. The dynamic algorithm places no restrictions on the number of DRs permitted, except limitation by the size of the packet header field used to store the packet's DR number.

The algorithms require only two virtual channels per physical channel for fully adaptive routing. One of the virtual channels is used for adaptive routing and the other for dimension-order routing. With two virtual channels per physical channels, there are a total of thirty-two 90° turns and eight 0° turns. The dynamic routing algorithm allows only twenty of the thirty-two 90° turns and four of the eight 0° turns.

The algorithms, by themselves do not guarantee that a packet will ever reach its destination. To guarantee progress toward a destination, misrouting is limited by placing an upper limit on the number of steps a message may take away from its destination. With this limit, a weighted sum of the distance to the destination and the number of misrouting steps remaining for all messages in the network is strictly decreasing. Thus the network is livelock-free. Both algorithms allow the network to gracefully degrade in the presence of faulty channels.

5.3.2 Backtracking Techniques

There is only one completely adaptive, nonminimal backtracking technique presented in the literature, which is described as follows.

The ***exhaustive misrouting backtracking*** (EMB) [45] scheme does a depth-first search of the network using both profitable and unprofitable links. It uses the "best first" heuristic, taking profitable links over unprofitable ones. Although, on the average, the established circuits are longer than a purely profitable scheme, EMB's probability of finding a path is greater.

The drawback of this algorithm is that it cannot detect that a message is undeliverable until it searches every path it can reach in the network. In the worst case, a single probe can tie up large amounts of network resources searching in vain for a nonexistent path.

5.4 Partially Adaptive, Minimal Routing

This section consists of the description of the following techniques: zenith algorithm, multiple path fixed routing, subcube hanging algorithm, buffer-reservation routing, extended up-preference routing, and K-family routing.

5.4.1 Progressive Techniques

Zenith Algorithm

This is a minimal, partially adaptive packet-switched routing technique for the binary hypercubes. It was presented in [71] and is deadlock, livelock, and starvation free. Messages are divided into two different classes: Class 1 and Class 2 messages.

Both Class 1 and Class 2 messages follow a minimal path from their source to their destination. Class 1 messages first turn the incorrect zero bits (in the history bit mask which keeps track of the dimensions traversed) into 1 bits (ascending phase) and then turn the incorrect 1 bits into 0 bits (descending phase). Class 2 messages do exactly the converse process. To guarantee deadlock-freedom, this algorithm requires that each node have three queues: Queue 0 holds only Class 1 messages during their ascending phase. Queue 1 holds Class 1 and Class 2 messages during their descending phase and Queue 2 holds Class 2 messages during their ascending phase.

Every message M is injected into the network as a Class 1 member in Queue 0 of the source node. It then starts moving from node to node through Queue 0 of the visited nodes. When the message arrives at $Zenith(M)$ (defined as $S \vee D$, where

S is the source and D is the destination), it is moved from Queue 0 to Queue 1 of $Zenith(M)$ and starts its descending phase moving through Queue 1 of the visited nodes. If at a certain node P, message M is at its ascending phase (i.e., climbing up to $Zenith(M)$) and it cannot make progress through any of the possible dimensions due to congestion problems, and if Queue 1 of that node has free space, then M is moved from Queue 0 to Queue 1 of node P. Therefore, M turn into a Class 2 message and starts its descending phase to $Nadir(M, P)$ (defined as $P \wedge D$, where D is the destination). Once it has arrived at $Nadir(M, P)$, it is moved from Queue 1 to Queue 2 of $Nadir(M, P)$ and it starts its ascending phase until it reaches the destination node D.

Multiple Path Fixed Routing

Li [86] presented a minimum set of deadlock-free routing restrictions for binary hypercube networks and a deadlock-free multiple-path fixed routing algorithm which provides a total of $k+1$ arc-disjoint paths between two nodes of distance k. Proofs are presented to show that any routing that complies with the above-mentioned restrictions is deadlock-free and that these restrictions are minimal in that any relaxation to them will cause deadlock. An important advantage of the routing restriction technique is that it does not require a message to be received entirely before it is forwarded. Therefore, it can be used in wormhole routing as well as in the store-and-forward routing. Routing restrictions mean the rules that specify which channels ending at a node can forward messages to certain channels starting from the node. Every routing algorithm implies a set of restrictions. The fixed routing algorithm allows a channel to forward messages only to channels that are at

a higher dimension. The multiple-path routing algorithm can be obtained by a small modification to the normal fixed routing algorithm. The minor modification, which allows increased communication capability, is that a message can be forwarded from a lower dimension channel to a higher dimension (which is the forwarding allowed in normal fixed routing), or from a higher dimension channel to a lower dimension channel if the latter is positive, where an ith dimension channel connecting nodes a and b is positive if the ith bit of a is 0 and that of b is 1 and the channel is directed from a to b.

Subcube Hanging Algorithm

Pifarré *et. al.* [111] proposed two routing algorithms for wormhole routing on the hypercube networks which are adaptive, deadlock- and livelock-free. The first one will be discussed in Section 5.5.1. The second algorithm is an adaptive, minimal algorithm, which we call the ***subcube hanging*** algorithm. This algorithm is mainly based upon considering the hypercube as a hierarchical network in which each node is a small hypercube. These small hypercubes are referred to as ***subcubes***. The routing functions over this network are defined as if the hypercube were ***hung*** from one of these subcubes. This hanging defines a leveled structure of subcubes. By considering the hierarchical network as hung from one of these subcubes, an order in which these subcubes are to be visited can be defined. Moreover, defining these subcube visits in a deadlock-free manner guarantees a deadlock-free routing algorithm on the whole. The strategy chosen for visiting the subcubes is as follows. The hierarchical hypercube is regarded as hung from one of the subcubes. The path of a message going from one subcube to another subcube consists of two phases. During the first

phase, the message visits the nodes of the hierarchical network by moving downward, considering the network as hung from a fixed subcube. During the second phase, the message visits the nodes by traveling upward. Each subcube may use a different "internal" routing strategy.

Buffer-Reservation Routing

Cypher & Gravano [24] focus on a particularly simple and important class of routing algorithms called the ***buffer-reservation*** algorithms for packet-switched networks. A buffer-reservation algorithm consists of rules that specify to which buffers a packet may move based solely on the buffer currently holding the packet, the packet's source node, and the packet's destination node. A packet is allowed to move from its current buffer to any other buffer at any time, provided that the other buffer is empty and that the move is allowed by the routing algorithm. Buffer-reservation algorithms can be implemented efficiently in hardware because they require only local information to make routing decisions, are inherently asynchronous and therefore do not require a global clock, and do not require the creation or exchange of any special packets containing only control information. The disadvantage of buffer-reservation techniques is that they require that each node contain some minimum number of buffers.

The paper mainly characterizes the properties which these algorithms must have in order to be free of deadlock and to use these properties to prove lower bounds on storage requirements. A simple example of an adaptive buffer-reservation algorithm is also presented which is deadlock free and requires no ordering of buffers.

Extended Up-Preference Routing

Mahmood, Lynch, & Shaffer [96] presented a class of adaptive, minimal deadlock-free routing algorithms for circuit-switched hypercubes. It is based on the following concept of transitions. Each link on the path changes a specific bit x_i in dimension i. If bit x_i changes from 0 to 1, then it is considered an ***up transition*** denoted U_i. Similarly, a ***down transition*** D_i changes x_i from 1 to 0. The ***UP Preference*** algorithm [22] specifies that any up transition U_i can be made at any point in the order of required transitions, but a down transition D_i can be made only after all lower dimension transitions have been completed. The ***Extended UP Preference*** algorithm uniformly applies the ***UP Preference*** rule to each of the 2D subcubes traversed in a source-destination path within a larger hypercube.

5.4.2 Backtracking Techniques

There is only one partially adaptive, minimal backtracking technique presented in the literature, which is described as follows.

K-Family routing was proposed by Grunwald & Reed [55] as an improvement over exhaustive profitable backtracking (see Section 5.2.2). Although the exhaustive backtracking scheme does not repeatedly search the same paths, it can visit a specific link or node several times. This can lead to unnecessary backtracking and longer setup times. The k-family routing scheme is a family of partially adaptive, profitable, backtracking scheme proposed for binary hypercubes that use a heuristic to help minimize redundancy in the search for a path. The techniques in the k-family scheme are two-phased, using a heuristic search in the first phase and an exhaustive search

in the second. Each technique is distinguished by a parameter k that determines when the heuristic is used and when exhaustive backtracking is used. When the header is at a distance greater that k from the destination, the heuristic is used. When the distance is less than or equal to k, an exhaustive profitable search is used. If $k = 1$, the scheme is a strictly heuristic search being exhaustive only for the last link. As k grows to the distance between the source and destination, the search becomes more and more exhaustive.

The k-family scheme makes use of a ***history mask*** contained in the circuit probe. At each level in the search tree, the cumulative history mask of the ancestor nodes determines which links might be explored. The history mask records all the dimensions explored at each link and all dimensions explored at all ancestor nodes. The heuristic limits exploration to dimensions not marked in the history mask. In this way the search tree is pruned of links that are likely to lead into areas of the network that have been searched previously. The number of possible paths provided by the scheme is $< n!$.

5.5 Partially Adaptive, Non-Minimal Routing

This section consists of the description of following four techniques: random routing, exchange model, inter- and intradimensional routing, n-phase dimension correction routing, and two-phase misrouting backtracking.

5.5.1 Progressive Techniques

Random Routing

This routing scheme was proposed by Valiant & Brebner [131, 132, 133]. Let N be a network with n nodes. Every link between nodes has a queue where it stores its message. For every pair of nodes a and b, a ticket τ_{ab} is computed with the route that should be traversed when going from a to b, e.g., τ_{ab} may be a list containing the path that should be traversed. The initialization consists of the choice of an intermediate random destination q for every packet p and the computation of $\tau_{aq}(p)$ and $\tau_{qb}(p)$. It is followed by two phases. In the first phase the message is sent from a to q. In the second phase, from q to b. There are two policies as to how to pass from the first phase to the second. The first one separates the two phases completely, i.e., at the end of phase 1, every packet is delivered to its intermediate destination, where it waits for the beginning of phase 2. Phase 2 starts when all packets have finished phase 1. The second policy allows every packet to begin its phase 2 immediately after it has finished its phase 1 route. The scheme is progressive, misrouting and partially adaptive and is applicable to a variety of topologies. The maximum path length for any transmission is equal to the diameter of the network.

Exchange Model

Exchange is a model presented by Ngai & Seitz for adaptive routing [103, 104]. Every node n_i has a predefined routing relation R_i that tells which of its neighbors is the next on the route of the packets currently in n_i with dimension n_j, for every n_j in the network. The transfer of packets between adjacent nodes is accomplished

via an *exchange* operation: if n_i has a packet p in a buffer b and n_j is selected to be the next node in the route of p, then any of the following cases may arise.

1. n_j has a packet p' in a buffer b' such that it also wants to exchange it with n_i. Then, they use the common link, p is allocated in b' and p' in b.

2. n_j does not want to exchange any packet with n_i. Then,

    a. n_j has an empty buffer b' that is not being used for another exchange operation. Then n_j receives p in b' and b receives the *null* packet, i.e., it is freed.

    b. n_j either has no empty buffer or all its empty buffers are being used to exchange through other links. Then, a buffer b' not currently being used for another exchange is chosen. The packet p' contained in b' is moved to b and p to b'.

The exchange is a nonminimal adaptive routing algorithm and is independent of the topology.

Inter- and Intradimensional Routing

Young & Yalamanchili [137] studied adaptive circuit-switched routing in the richly connected generalized hypercube architecture. Three adaptive routing techniques were proposed for this architecture: *interdimensional routing*, *intradimensional routing*, and *intra/interdimensional routing*. These protocols are all partially adaptive, misrouting, and progressive.

Intradimensional routing transmits the header one dimension at a time. When the link needed to traverse a dimension is not available, the header is misrouted over another link in the same dimension. To avoid unnecessary long paths, a time-out interval defines when path setup is aborted, and transmission is retried later. This time-out interval is typically set to 10 times the time it takes to cross one link (that is, a maximum path length of 10). This abort/retry mechanism avoids deadlock, while the time-out interval prevents livelock.

Interdimensional routing also attempts to traverse dimensions one at a time in the same order as e-cube. If the required profitable link is busy, the header is misrouted over a link in another dimension. To avoid cycles, a header is restricted to visiting a node only once. Again, a time-out interval aborts path setup and schedules a retransmission later.

Intra/interdimensional routing combines the two techniques in an attempt to get the best features from each. Again, the technique attempts to traverse dimensions in the same order as e-cube. If the profitable link is unavailable, the scheme will attempt to perform intradimensional misrouting. If this fails, interdimensional misrouting is attempted.

n-phase Dimension Correction

Pifarré *et. al.* [111] proposed two routing algorithms for wormhole routing on the hypercube networks which are adaptive, deadlock- and livelock-free. The first of these which we call the ***n-phase dimension correction*** algorithm is a nonminimal algorithm. In this technique the route of a message consists of n phases. Each phase processes one dimension. Dimensions are processed from highest to lowest in an

ordered manner. Each dimension has to be processed, regardless of whether it is already correct. During each of these phases, every message will have d alternative paths to take for some d fixed in advance. The main idea behind this algorithm is to break the structured communication patterns by providing a number of paths between any source and destination, introducing a limited potential of derouting at each phase. At the start of phase i, dimensions $n-1$ through $i+1$ have already been corrected and will never be modified again. When dimension i is being processed, a message m will either have to correct it or not. Regardless of this need, m will be sent first through a dimension $j < i$ (if i is not one of the least significant dimensions), regardless of whether this means correcting dimension j. Only after this will the message be sent through dimension i, if required. Dimension j will be chosen from within a set of d dimensions. In principle, the set of d dimensions corresponding to a certain phase of the algorithm may be different for each node of the network, but it must be fixed in advance. In other words, if a message m is at some node p, m is at the ith phase of the routing algorithm, then m will leave p through any of the dimensions that node p will have associated with the ith phase. The second algorithm has already been discussed in Section 5.4.1.

5.5.2 Backtracking Techniques

There is only one partially adaptive, nonminimal backtracking technique presented in the literature, which is described as follows.

To improve on the exhaustive misrouting backtracking scheme (see Section 5.3.2), a partially adaptive scheme analogous to the k-family (see Section 5.4.2) scheme

was proposed [46], where the probe is free to misroute only if it is within a certain distance u of its destination. The first phase of two-phase backtracking (TPB-u) performs an exhaustive profitable search when the probe is at a distance greater than u from the destination. When TPB-u is within u of the destination, it enters the second phase, where it performs an exhaustive misrouting search. A TPB-u probe can switch phases several times during a single route.

What has been presented above can be summarized in a diagrammatical form as shown in Figures 5.1, 5.2, 5.3, and 5.4. The trees in these figures have been organized based on topology and follow the classification proposed in Chapter 3. It can be seen that most of the work done so has concentrated on the hypercube topology. That is why there exist routing techniques in each class. However, it is also obvious that there is only scarce amount of literature on backtracking based schemes. The k-ary n-cube, and mesh topologies have been rather overlooked in the past. That is why we only find a small number of proposed techniques. Moreover, not all branches of the classification tree Figure 3.1 have been explored as is evident from Figures 5.2, and 5.3.

5.6 Comparison of One-to-One Routing Techniques

A comparative study of various routing techniques for hypercube networks is presented below, which is also summarized in Table 5.1 and 5.2. Most of these techniques are deadlock- and livelock-free, except where explicitly mentioned.

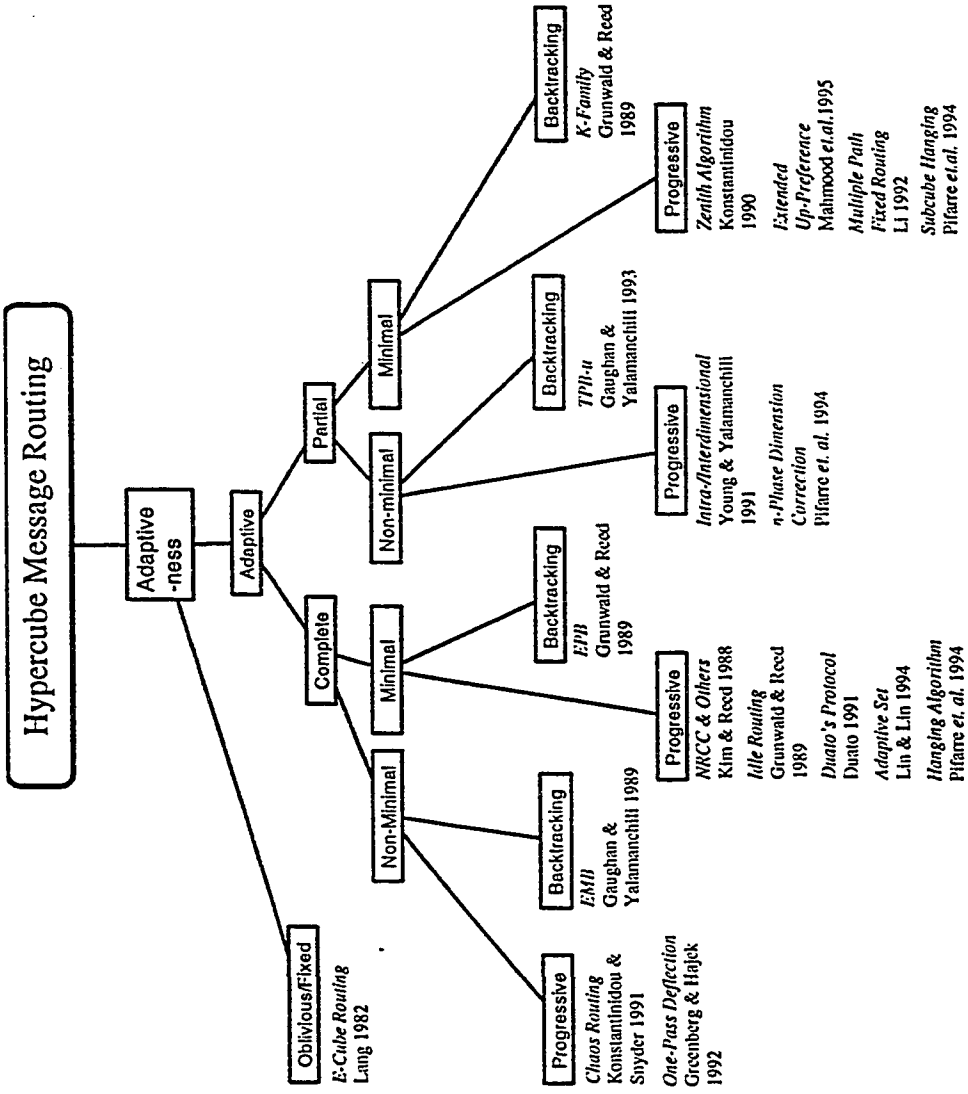


Figure 5.1: Classification of hypercube routing techniques.

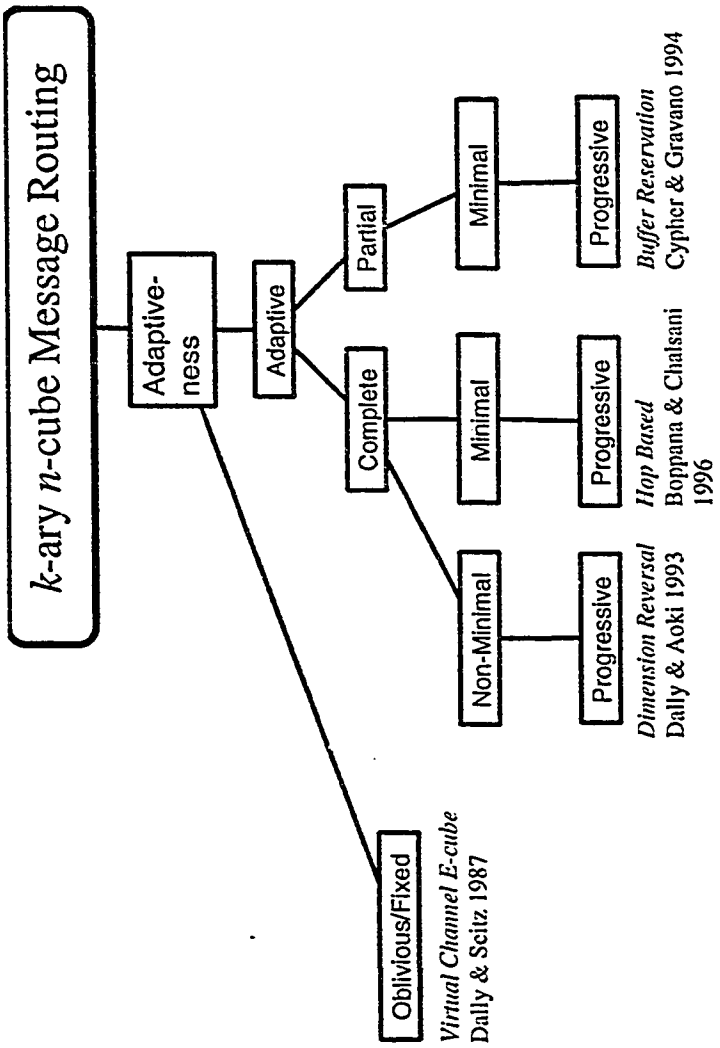


Figure 5.2: Classification of k-ary n-cube routing techniques.

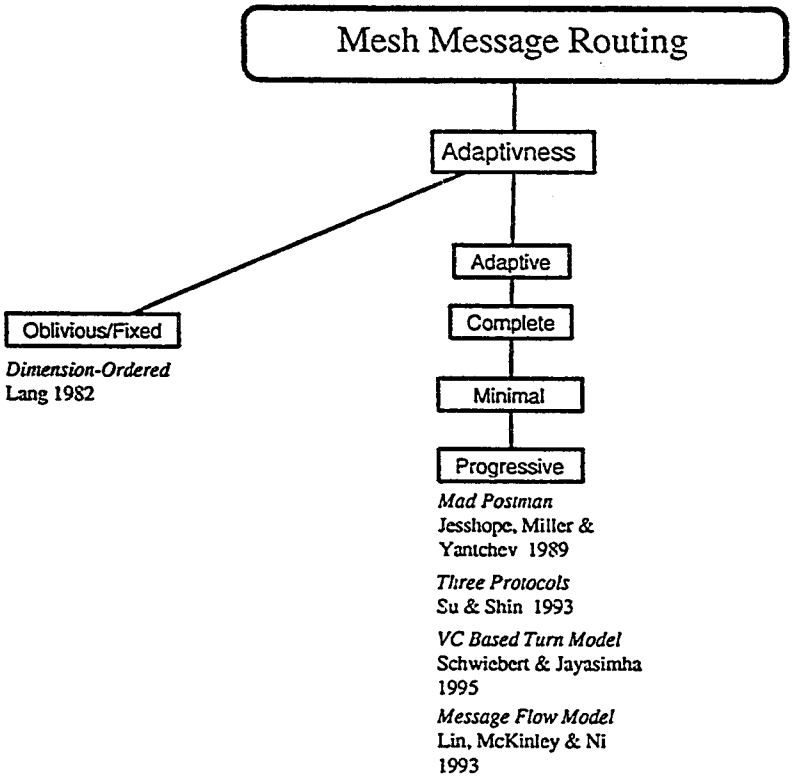


Figure 5.3: Classification of mesh routing techniques.

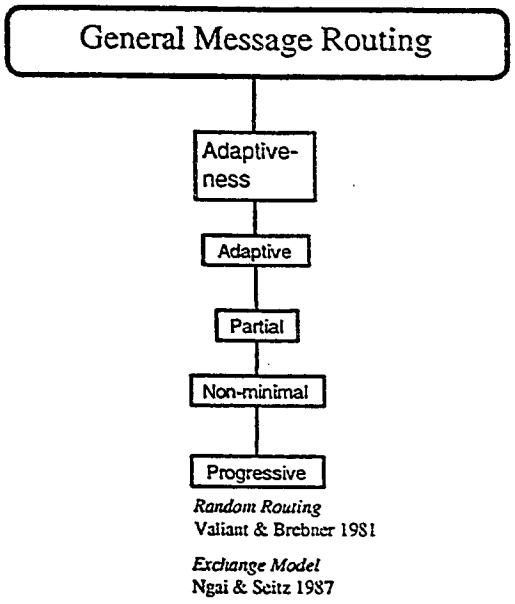


Figure 5.4: Classification of general routing techniques.

In the *intra-/inter-dimensional* technique [137] the timeouts used for avoiding long paths lead to excessive delay. In *Duato's protocol* [38] as the message gets closer to its destination, the number of alternative paths is reduced dynamically. *Idle* routing [55] may abort an almost completely established path, wasting bandwidth, in order to prevent deadlock. In *exhaustive profitable backtracking* (EPB) the exhaustive search of all paths leads to longer set-up time and the history information kept in the routing header for backtracking is an overhead. *K-family* [55] minimizes redundancy in the search for a path, but adds an overhead of history mask. Moreover, the heuristic parameter k is difficult to determine *a priori*. In *exhaustive misrouting backtracking* (EMB) [46] the exhaustive nature leads to longer set-up time. Moreover, it cannot detect that a message is undeliverable until it searches every path it can reach tying up a lit of resources. In *two-phase backtracking* (TPB) technique

[46] the parameter u, in general, is difficult to determine. Moreover, it is exhaustive in nature which leads to excessive set-up time. The *zenith* algorithm [71] may cause hotspots at nodes 0 and $2^n - 1$ of the hypercube. In *chaos* routing, [72, 73, 74, 75] livelock may be caused due to random misrouting. Also, randomly choosing a message to misroute may cause a message to starve. The *hanging* technique [111] may cause hotspots at certain nodes. In *NRCC* and other related techniques, [67] the use of global information causes a lot of overhead. Also, centralized routing causes a lot of delay and overhead. *One-pass deflection* [54] is deadlock-free but not livelock-free. The cause of livelock is random misrouting. Its computational complexity is $O(d/2)$, where d is the number of packets transmitted by a node per unit of time. The maximum number of misroutes a packet suffers is $O(log\ d)$.

The comparison of various routing techniques for k-ary n-cube networks is summarized in Table 5.3. All of these techniques are deadlock- and livelock-free

The comparison of various routing techniques for mesh networks is summarized in Table 5.4. All of these techniques are deadlock- and livelock-free.

A comparison of various routing techniques for general networks is summarized in Table 5.5. All of these techniques are deadlock- and livelock-free

5.7 Summary

In this chapter we have discussed the various routing techniques for one-to-one or unicast communication. Different characteristics of each technique have been given. Also, a brief discussion of the operation of the schemes is described. The techniques have been organized according to the classification presents earlier. Moreover, a

Table 5.1: Comparison of Hypercube Routing Techniques

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Intra/Inter Dimension	CS	Partial	Non-Minimal	Prog.	NA
Duato's Protocol	SF WH	Complete	Minimal	Prog.	Any no. of VCs necessary
E-cube	SF, CS VCT, WH	Oblivious			NA
Idle	CS	Complete	Minimal	Prog.	NA
EPB	CS	Complete	Minimal	Back.	NA
K-Family	CS	Partial	Minimal	Back.	NA
EMB	CS	Complete	Non-Minimal	Back.	NA
TPB-u	CS	Partial	Non-Minimal	Back.	NA
Zenith	SF	Partial	Minimal	Prog.	3VC/PC

Table 5.2: Comparison of Hypercube Routing Techniques, Continued...

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Chaos	SF	Complete	Non-Minimal	Prog.	$log\ n$ n: no. of nodes
Message Flow	WH	Completely	Minimal Non-Minimal	Prog.	2 VC/PC
Adaptive Set	WH	Complete	Minimal	Prog.	$\lceil (n+1)/2 \rceil$ VC/PC
n-Phase Dim. Correction	WH	Partial	Non-Minimal	Prog.	8 VC/PC
Hanging	SF	Complete	Minimal	Prog.	NA
Subcube Hanging	WH	Partial	Minimal	Prog.	8 VC/PC
NRCC & Others	SF	Complete Partial	Minimal Non-Minimal	Prog.	NA
One-Pass Deflection	SF	Complete	Non-Minimal	Prog.	NA
Extended Up-Pref	CS	Partial	Minimal	Prog.	NA
Multiple path fixed	WH SF	Partial	Minimal	Prog.	NA

Table 5.3: Comparison of k-ary n-cube Routing Techniques

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Hop Based	WH	Complete	Minimal	Prog.	$n\lfloor k/2 \rfloor$ for + hop $\lceil n\lfloor k/2 \rfloor /2 \rceil + 1$ for - hop
Dimension Reversal	WH	Complete	Non-Minimal	Prog.	2 VC/PC
VC Based E-cube	WH	Oblivious			2 VC/PC
Buffer Reservation	SF	Partial	Minimal	Prog.	NA

Table 5.4: Comparison of Mesh Routing Techniques

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Dim. Ordered	SF, CS VCT, WH	Oblivious			NA
Mad Postman	Modified WH	Complete	Minimal	Prog.	2 VC/PC
Three Protocols	WH	Partial	Non-Minimal	Prog.	2 VC/PC
Opt-y	WH	Complete	Minimal	Prog.	2 VC/PC

Table 5.5: Comparison of General Techniques

Technique	Switch. Tech.	Topology	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Random Routing	SF	Any Topology	Partial	Non-Minimal	Prog	NA
Exchange Model	SF	Any Topology	Partial	Non-Minimal	Prog	At least 2 VC/PC

comparison has been presented between the various one-to-one routing techniques. The comparison has been based on the classification presented earlier together with the topology and other features such as the complexity, the size of the routing path, etc.

Chapter 6

Fault-Tolerant One-to-One Routing

The ***fault-tolerance*** of an interconnection network is a measure of its ability to provide (at least a degraded level of) service, even in the presence of faults in the components that constitute the network. If the system performance degrades gracefully, in the presence of faults, the system is said to be ***fault-tolerant.*** In static interconnection networks, there are two network components: communication links or simply ***links*** and processing elements or ***nodes.*** A fault in either of these components can be ***permanent*** or ***transient.***

The reliability $R(t)$ of an interconnection network is a function of time, defined as the conditional probability that the interconnection network will operate correctly throughout the interval $[t_0, t]$, given that it was operating correctly at time t_0. In other words, the reliability is the probability that the interconnection network will operate correctly through a complete interval of time.

Fault-tolerance is a technique that can improve reliability, but a fault-tolerant system does not necessarily have to have a high reliability. The reliability of a system also depends on individual component reliabilities. If the same "quality" components are used in the entire system, the fault-tolerance schemes increase overall system reliability. On the other hand, a very simple system can be built with extremely "good" components such that the probability of system failure is very low. In other words, a system can achieve a high reliability but not possess the attribute of fault-tolerance.

Fault-tolerance is measured in terms of ***network connectivity.*** Network connectivity measures the ability of a network to continue operation despite failed network components (i.e., nodes and links). Informally, network connectivity is the maximum number of network components that must fail to partition the network into disjoint subnetworks. This leads to ***link connectivity*** of a network which is defined as the minimum number of links that must be removed from the network in order to partition the network into subnetworks. The node connectivity is defined analogously. Fault-tolerance may be modeled by the following components:

- ***Fault Model:*** This component of fault-tolerance model specifies what type of faults are anticipated. The fault model does not have to correspond exactly to the real world problems that could develop in the components. The model may be chosen to simplify reliability analysis.

  In multicomputer networks, faults can be of two types, i.e., ***node fault*** and ***link fault.*** A third hybrid type, the ***block fault***, arises from a combination of both types of failed components to form collections of faulty regions called

fault blocks. Another type of fault, sometimes referred to as ***transmission fault*** occurs when any particular message transmission fails so that, although a message is sent along a link, it is not received at the other end. Such failures are usually ***short-duration***, i.e., if two messages are sent along the same link at different time instants, a single failure cannot prevent both messages from arriving at the other end.

- ***Fault-Tolerance Criterion:*** This represents the minimal service that is to be expected from the interconnection network which is known to have certain faults. In other words, this is the basis on which it is judged whether the interconnection network in question has tolerated the faults in a reasonable manner. The more stringent the criterion becomes, the more elaborate the design of the interconnection network becomes.

  Fault-tolerance may be classified as ***global network information*** based or ***local network information*** based, according to whether the information regarding faulty components is distributed in the entire network or each node knows only its adjacent faulty components. In global network information model each fault-free node knows all faults in the system, whereas in local network information model each fault-free node knows the status of its neighbors only. The global-knowledge model requires some form of routing tables, which are unnecessary for fault-free routing. Distributing fault information to each and every node in a massively parallel processor is expensive. Also, routing algorithms that depend on global fault information should have alternate schemes to transmit fault status messages during the ***transition*** period —

the interval between the occurrence of a fault and the time at which this fault is known globally.

The ideal fault-tolerance criterion is that the network performs exactly the same functions it performed before the faults were developed. Of course, it can be seen that this ideal criterion cannot be met without investing a lot in extra hardware and/or software. In practice, a criterion weaker than this is usually specified.

Fault-tolerance can be achieved by many techniques. One approach that is often used in context of interconnection networks is to detect and locate the fault that has occurred and reconfigure the network to remove the faulty component. This implies that the network should have some form of redundancy. This redundancy can take one of several forms.

- ***Hardware redundancy*** is the addition of extra hardware to tolerate faults. For example, extra links or nodes may be added in a network to tolerate link or node failures.

- ***Software redundancy*** is the addition of software to detect and possibly tolerate faults. For example, in static interconnection networks, this might take the form of an improved routing algorithm that dynamically reroutes a message if a link and/or node failure has occurred.

- ***Information redundancy*** is the addition of extra information beyond that required to implement a given function. For example, error detecting codes use a form of information redundancy.

The following are some important definitions with respect to fault-tolerant routing [48]:

1. A network is said to be ***connected with respect to a routing protocol*** if the protocol can route a message between any pair of nonfaulty nodes.

2. A protocol is said to be ***f fault-tolerant*** if for any f faults in the network, the network is still connected with respect to the protocol.

3. A protocol is said to be ***f fault-recoverable*** if for any f faults or less in the network, a message that is undeliverable will not hold the network resources indefinitely. Thus, if a network is fault-recoverable the faults will not induce deadlock.

In ideal situations, the network would be f fault-tolerant for very large f. Practically, however, a system may be satisfactory, which is f fault-recoverable so that only the functionality of a small part of the network suffers as a result of a local disconnection. Certainly, such situations need to be avoided where a few faults may cause a catastrophic failure of the entire network.

The following theorems have been proposed in the context of k-ary n-cubes with $n > 1$ and $k > 2$. The proofs may be found in [48]:

Theorem 1: No profitable protocol is 1 fault-tolerant.

Theorem 2: A misrouting backtracking protocol with 1 misroute is 1 fault-tolerant.

Theorem 3: If $2n$ is the degree of each node in a network, then a misrouting backtracking protocol with 3 misroutes is $2n - 1$ fault-tolerant. That is, in order to achieve fault-tolerance up to the degree of the node minus 1, only 3 misroutes are necessary regardless of node degree.

An important observation is that wormhole routing messages are nonabortive. Once the header leaves the source node, the messages will tie up network resources until it is successfully routed to its destination. If faults stop its progress, it will stall in the network. Backtracking, on the other hand, is abortive. If a circuit cannot be established, the probe will eventually backtrack to the source node. Thus, backtracking protocols, as a class are f fault-recoverable for any f. Wormhole routing protocols (which are progressive by definition) are not fault-recoverable.

A study of fault-tolerant routing techniques for one-to-one communication is presented. They are organized according to the classification proposed in Chapter 3. A comparison of the techniques presented here is also presented in a tabular form at the end of the chapter, where the techniques are compared on the basis of the proposed classification. Moreover, a comparison is also made based on the fault-tolerance properties of the schemes.

6.1 Completely Adaptive, Minimal Routing

This section presents the following techniques: subcube-based hanging, exchange based safety routing, multiple virtual network routing, compressionless routing, and table-based routing.

Progressive Techniques

Subcube-Based Hanging

Park [110] proposed a fault-tolerant wormhole routing algorithm in an n-dimensional hypercube in the presence of up to $n-1$ faults. It uses the virtual channel concept in order to provide both deadlock-freedom and adaptiveness. The proposed algorithm divides n-dimensional hypercube into smaller subcubes so that perfect subcubes can be utilized to relieve the communication difficulties in faulty subcubes. After the division of the hypercube into two subcubes, two cases may occur: at least one of the subcubes is perfect, or both subcubes contain some faulty nodes.

In the first case, routing can take place in the nonfaulty subcube in a deadlock-free manner whenever needed. In the second case, the hypercube is further subdivided into subcubes of smaller sizes until a perfect subcube is obtained. The hypercube is then hung from this subcube and the hanging algorithm of [111] is applied. The total number of virtual channels used is $2\lfloor\frac{(n-1)}{(n-k)}\rfloor+3+|K'|$, where k is such that $k \leq n-2$ and for every $n-k$ dimensional hypercube produced from the n-dimensional hypercube has at least one of $n-k-1$ dimensional perfect hypercubes. $|K'|$ is the number of intrasubcubal links. Maximum path length is $\lfloor\frac{n-1}{n-k}\rfloor+p$, where p is the Hamming distance between the source and destination.

Exchange Based Safety Routing

Chiu & Wu [21] developed a routing strategy which adopts a technique similar in nature to the one proposed in [84] for hypercubes. The concept of unsafe nodes and its extension are used in this scheme. A set of strict criteria was presented to

identify the possibly bad candidates, called ***unsafe nodes***, to forward a message. As a result, the number of such undesirable nodes is reduced without sacrificing the functionality of the mechanism. Various properties of unsafe nodes are studied in detail. Rigorous proofs have been provided to these properties. Another notion of the ***degree of unsafeness*** for classifying the unsafe nodes was introduced. The algorithm is guaranteed to achieve a path length of no more than the Hamming distance between the source and destination plus four, if the number of faulty nodes is no more than $n-1$. Deadlock-freedom is achieved with only a constant number (5) of virtual networks. Deadlock-freedom is ensured by restricting the use of virtual channels in increasing order of network numbers, and having up-transition traversals be considered over down-transition ones. Maximum path length is $p+2$, where p is the Hamming distance between the source and destination.

Multiple Virtual Network Routing

A theoretical model, based on the virtual channel network, for the study of fault-tolerance in k-ary n-cubes has been developed by Linder & Harden [94], which we call the ***multiple virtual network*** scheme. Their proposal is based on the concept of virtual network. Virtual networks are mapped onto a physical network by splitting physical channels into virtual channels. This is an extension of the concept of virtual channels to multiple virtual interconnection networks that provide adaptivity, deadlock-freedom and fault-tolerance. Each physical link is shared by many virtual channels. These virtual channels can be divided into several groups or virtual networks. The number of virtual channels used depends on how many virtual networks are needed. Message passing inside a virtual network is deadlock-

free and messages are constrained to travel through virtual networks in a defined order. When the message is blocked in a virtual network, it can keep going forward via another virtual network, thereby increasing routing adaptability. The chief disadvantage of this method is that many virtual channels (in general an exponential function of the network dimension) are required, e.g., a k-ary n-cube needs $2^{n-1}(n+1)$ virtual channels, thus it is feasible only for low dimensional networks. Moreover this scheme can tolerate only one faulty node. It also does not make use of locality of communication, breaking a bi-directional k-ary n-cube into two virtual interconnection networks.

Compressionless Routing

Compressionless Routing (CR), proposed by Kim, Liu & Chien [70], is a framework which supports both adaptive and fault-tolerant routing for a wide variety of network topologies (specifically k-ary n-cube) and requires no virtual channels for deadlock prevention. ***Fault-tolerant Compressionless Routing*** (FCR) extends CR to support end-to-end fault-tolerant delivery. It has the advantages that it tolerates transient faults while maintaining data integrity, tolerates permanent faults, can be applied to a wide variety of network topologies and eliminates the need for software message buffering for reliability.

The basic idea in FCR is to use the retransmission mechanism to tolerate transient faults and use unrestricted routing flexibility to circumvent permanent network faults. When a transient fault is detected, the detecting router sends kill signals both forward and backward along the message path. If it receives a backward kill signal, a source node retransmits the same message. In case part of the message has already

arrived at the destination, it is canceled by the forward kill signal. When a message encounters a permanent fault, it circumvents it by using alternate paths.

To ensure reliable transmission, FCR requires that each message holds its path until the last data flit has reached its destination. Successful delivery of a message is not guaranteed until the message's entire data portion has been delivered to the destination without error. Thus, the message sender must hold the tail of the message until the last data flit is delivered. Messages killed due to faults are retransmitted by the source node. No path history is retained, so it is also possible that the retransmitted message will use the same path. If the fault was transient, the message may complete transmission the second time. If the channel is permanently faulty, the message is killed again and retransmitted. After a number of errors, the channel is marked as permanently faulty. Retransmission and multipath routing allow FCR networks to tolerate most permanent faults. However, not all messages have multiple paths in a minimal, adaptive router. If a message with a unique path to its destination is blocked by a permanent fault, retransmission will not solve the problem. To tolerate these faults, FCR uses disciplined misrouting. The maximum number of misroutings that a message may take is predetermined, allowing source nodes to detect if the message has completed transmission. Thus the algorithm can tolerate a large number of random faults.

Table Based Routing

This routing technique, presented by Reddy & Freitas [114], is at one of the extremes in adaptivity, i.e., it uses only two alternate routes between any source and destination. In systems employing table based routing, each node contains a

table of preferred directions for sending a message to each destination. There are several approaches for providing alternate paths to avoid congestion. One of the approaches is that whenever a message gets blocked, it frees up the established path. An attempt is then made to route the message to the destination through an alternate path. This requires that the message be sent to an intermediate destination from which the message will be forwarded to the correct destination. A table of intermediate destinations at each source is created for each destination.

Routing tables are computed by finding the shortest path between each pair of nodes. When there are several shortest paths, the source picks one of the outgoing directions of these paths randomly. After this computation, this preferred direction for each pair is entered into the source routing table. Once all the routing tables are computed, the intermediate node addresses are computed as follows. The shortest path between a pair of nodes is removed from the interconnection, and a new shortest path between the two nodes is computed based on the already computed routing table. The computational complexity of the scheme is $O(n^3)$.

6.2 Completely Adaptive, Non-Minimal Routing

This section presents the following techniques: sidetrack routing, history based routing, reduced overhead history based routing, cluster based routing, convex set routing, node labeling technique, depth-first search based routing, best first search routing, and misrouting backtracking.

6.2.1 Progressive Techniques

Sidetrack Routing

Gordon & Stout [53] considered an approach called "sidetrack" to route messages in faulty hypercubes. Assume that each node stores the fault status on the neighboring nodes. A message is sent in the forward direction via a randomly chosen link which is connected to a nonfaulty neighbor. When no such link is available, i.e. a message is blocked, the message is sent to a randomly chosen nonfaulty neighbor. It has been shown empirically that the probability of successful delivery of a message is high and approaches one when n, the dimension of the faulty hypercube, is large. This method can also be applied to a wide variety of topologies such as meshes, tori and other graphs. However, it also has some disadvantages. First, although the probability is low, routing failure does occur. Second, excessive message delay may arise even when few faults are present since little information is used to guide the messages. Moreover, the scheme is designed for store-and-forward type of switching and is not applicable to wormhole-routed networks.

History Based Routing

History based routing consists of routing messages between nodes based on a dynamic data structure (usually a stack) which keeps track of which nodes have already been visited. A history based routing technique has been designed by Chen & Shin [16, 18], which requires each node to know only the condition of its own links. This scheme attempts to route every message to its destination via an optimal path. When the insufficient knowledge on faulty components causes a message to be sent

to an intermediate node from which there is no optimal path to the destination node, an alternative path is chosen in such a way that the connectivity of the hypercube is fully exploited, and those faulty components encountered before will not be encountered again in future. This scheme is capable of routing messages between any pair of nonfaulty nodes as long as the total number of faulty components is less than n in an n-dimensional hypercube. Due to the absence of information on all faulty components in the network, the paths chosen by this scheme may not be the shortest. The maximum path length is $O(log_2\ n)$.

In this scheme, to indicate the destination of a message, the coordinate sequence (an ordered list of dimensions to be traversed to reach the destination) of a path $[c_1, c_2, \ldots, c_n]$ is sent along with the message. Additionally, each message is accompanied by an n-bit vector $tag = d_n d_{n-1} \ldots d_1$ which keeps track of spare dimensions that have been used to misroute in order to bypass faulty components. Therefore, a message can be represented as $(k, [c_1, c_2, \ldots, c_k], message, tag)$, where k is the length of the remaining portion of the path and is updated as the message travels towards the destination. When a node receives a message, it will check the value of k to see if the node is the destination of the message. If not, the node will try to send the message along one of those dimensions specified in the remaining coordinate sequence. The coordinate sequence will also be updated as the message travels through the hypercube. Each node will attempt to route messages via shortest paths first. However, if all the links in those dimensions leading to shortest paths are faulty, the node will use a different dimension to misroute the message via an alternative path.

Reduced Overhead History Based Routing

Lan [78] presented a fault-tolerant routing algorithm with some modifications to the algorithm proposed in [16] reducing the message overhead for hypercubes. It uses the concept of ***preferred dimensions*** and ***spare dimensions.*** If the distance between node u_i and node u_j is d, then $u_i \oplus u_j$ has value 1 at d bit positions, corresponding to d distinct dimensions. These d dimensions are called ***preferred dimensions.*** The remaining $n - d$ dimensions are called ***spare dimensions.*** The term preferred dimension is a relative term. By default, it means a preferred dimension at the local node with respect to the destination node.

Realizing that the preferred dimension along which a message has to travel from one node to another node is contained in the exclusive-OR value of their addresses, a message needs to carry the destination address only, instead of the sequence of coordinates, $[c_1, c_2, \ldots, c_n]$ and the number of coordinates in the sequence (k). Without $C[k]$ and k, looping may occur: when a spare dimension h is selected at node u_1 to pass a message to node u_2, dimension h becomes a preferred dimension at node u_2. It is possible that u_2 could send the message back to u_1, although other nonfaulty preferred dimensions may exist at node u_2. To prevent this situation from happening without $C[k]$ and k in the message, it has been suggested that a lower priority be assigned to the incoming dimension when selecting an outgoing preferred dimension. The incoming dimension is selected only when all other preferred dimensions are faulty. The only message overhead required by the algorithm is the destination address and a tag (an n-bit binary vector). The algorithm guarantees the delivery of messages from a source node to any destination, provided that the total number

of faults is less than the dimension of the hypercube. The maximum path length is $O(log_2\ n)$.

Cluster Based Routing

Gu & Peng [56] present an algorithm for node-to-node cluster fault tolerant routing in n-dimensional hypercubes. A cluster of a graph G is a connected subgraph of G and a cluster is called faulty if all of its nodes are faulty. In ***cluster fault tolerant routing***, the number of faulty clusters and the size of the clusters that an interconnection network can tolerate are studied. The authors propose an algorithm which, given at most $n-1$ faulty clusters of diameter at most 1 with at most $2n-3$ faulty nodes in total and nonfaulty source and destination nodes, finds a fault-free path from the source to the destination of length at most $n+2$. This technique is a natural extension of conventional node fault tolerant routing model, in which individual nodes are considered as faulty instead of clusters. The algorithm partitions the hypercube into two subcubes, each containing the source and the destination respectively. If either of the subcubes contains at most half of the faulty nodes, then the algorithm tries to find a fault-free path to an intermediate node in the subcube containing at most half of the faulty nodes. The above process is applied again reducing the dimension of the subcubes until a complete path is established. Maximum path length is $n+2$ and the computational complexity of the scheme is $O(n)$, where n is the number of dimensions.

Convex Set Routing

Ngai & Seitz have proposed a framework for adaptive routing in multicomputer networks [105], [106] for k-ary n-cube and mesh topologies. It is based on an adaptive virtual cut-through switching technique in the sense that the message packets are forwarded in a virtual cut-through style. However, the technique avoids the necessity of having to block, i.e., hold a packet and wait until the requested resource becomes available and has been acknowledged. It enforces the rule that all communicating processors agree to always accept a request from neighbors through preemption of the requests resource if necessary. Data transfer is thus guaranteed locally between every pair of adjacent nodes. This is called a ***coherent transfer protocol*** and is proved to be deadlock-, livelock- and starvation-free.

Using the coherent transfer protocol, a fault-tolerant routing technique has been proposed which is based on the following two notions:

- The non-faulty nodes and channels are referred to as the ***survived nodes*** and ***survived channels.*** Given a faulty network, the set of all survived nodes and survived channels together constitute a ***survived subnet.***

- Given the set R of routing relations of a non-faulty network and set of nodes N, a set of survived nodes $S \subseteq N$ is ***convex*** under R if for every pair of nodes $n_i, n_j \in S$, there exists at least one legal route from n_i to n_j that is guaranteed by R.

Thus when the entire set of survived nodes and channels of the network form a convex set, all the survived nodes can still continue to communicate with each other.

These communications are achieved with virtually no change in the basic routing decision mechanism at each node. The only added requirement is for a router to recognize its own faulty channels.

When the set of survived nodes and channels does not form a convex set under R, because certain pairs of nodes do not have legal routes joining them that are consistent with R, the following has been suggested and investigated: selectively discard certain survived nodes that are particularly difficult to communicate with, so that the remaining subset of survived nodes constitutes a convex set.

Node Labeling Technique

This technique uses a node labeling scheme that identifies nodes which cause routing difficulties. The technique, proposed by Boura & Das [13], is based on the node fault model and the use of virtual channels. Link faults are treated as node faults. Faulty regions are converted to rectangular regions to facilitate routing decisions at each node. The node labeling strategy is based on the following two rules:

1. ***Node Deactivation Rule***: A ***healthy*** node connected to 2 ***faulty*** nodes is deactivated and marked ***faulty***.

2. ***Node Activation Rule***: A deactivated node connected to a ***healthy*** node is activated and marked ***unsafe***.

First, faulty regions are transformed to rectangular regions by applying the ***node deactivation rule***. Nodes that present potential routing problems are deactivated. Second, the ***node activation rule*** is applied to activate deactivated nodes residing at the boundaries of faulty regions.

Messages are routed adaptively in healthy regions of the network. Once a message faces a faulty region, it is routed around it using a non-minimal path to get closer to its destination. The virtual channels are divided into various classes, two of which is used for transmission from a healthy node to another healthy node and to avoid deadlock. The remaining third class of virtual channels is used to route messages around faulty regions. Also a channel may be positive or negative as it directs the message in the positive or negative direction.

6.2.2 Backtracking Techniques

Depth-First Search Based Routing

This technique was studied by Chen & Shin [17] and every node is required just to know the condition of its adjacent components. To indicate the destination of a message, the relative address of the destination node is sent along with the message. The depth-first search (DFS) routing algorithm attempts to avoid visiting the same node more than once except when backtracking is forced. Thus, those dimensions that a message traversed so far are recorded in a set TD in the same order as visited, and will be delivered together with the message to the next node. Using TD^R, each intermediate node can determine the addresses of those nodes visited before, where TD^R is the reverse of TD. Clearly, sending TD along with the message is much cheaper than sending the addresses of all the nodes visited. The information to be passed on to the next node can be represented as $(d, message, TD)$, where d is the relative address of the destination node with respect to an intermediate node, and is updated as the message is routed toward the destination. When a node receives

a message, it will check the value of d to see if the destination is reached. If not, the intermediate node will try to send the message along an optimal path to the destination. However, if all the optimal paths are blocked by faulty components and those nodes visited before, the node will route the message via an alternative path using the concept of depth-first search. When there is no alternative path available, backtracking is enforced. An intermediate path can determine whether or not its ith link is connected to a node that was visited before. When backtracking is forced, the message must be returned to the node from which this message was originally received. Computational complexity of the technique is $O(n)$.

Best First Search Routing

Blough & Wang [7] consider the problem of fault-tolerant routing in multicomputers when incomplete, or partial, diagnostic information is available. A type of partial diagnosis known as ***k-reachability diagnosis*** has been defined. The overhead for k-reachability diagnosis increases with k, which specifies the radius of diagnostic information maintained by each node. A fault-tolerant routing algorithm has also been proposed which makes use of k-reachability diagnostic information and which allows a trade-off between the amount of diagnostic information and the quality of routing. The algorithm can handle an arbitrary topology. The technique assumes that each node initially knows the system topology but does not know the fault status of any other node.

In k-neighborhood diagnostic, each nonfaulty node determines the status of every node from which its distance is k or less. Thus, 1-neighborhood diagnosis consists of each nonfaulty node determining the status of its neighbors. For a system with

diameter d, d-neighborhood diagnosis corresponds to each nonfaulty node having complete diagnostic information. For $1 < k < d$, k-neighborhood diagnostic provides nonfaulty nodes with a level of diagnostic information between the two extremes. A node j is k-reachable from a node i if there exists a nonfaulty path of length at most $k-1$ from i to a neighbor of j. In k-reachability diagnosis, each nonfaulty node determines the status of every node which is k-reachable from it.

The routing algorithm uses a best first search approach and it is completely adaptive, misrouting and backtracking. It starts with the source node computing a shortest feasible path to the destination. The source node then begins to forward the message along this path. As the message is forwarded, each intermediate node checks to verify that none of the nodes in the path are known to be faulty. As long as this condition is met, the message is forwarded along the initial path and each intermediate node is marked as visited. At some point, an intermediate node may find that one of the nodes in the path is faulty. This means that the message must be rerouted. The intermediate node then attempts to find a new shortest feasible path to the destination. If such a path exists, the intermediate node forwards the message along this new path. If no such path exists, then the procedure backtracks by sending the message back along the current path and attempting to find new feasible paths to the destination. If this backtracking procedure leads to the source node and no more feasible paths from the source exist, the message is not deliverable. Maximum path length is $2n-k-3$.

Misrouting Backtracking

Gaughan & Yalamanchili [48] presented a class of deadlock-free adaptive routing algorithms called ***misrouting backtracking with m misroutes*** (***MB-m***) for k-ary n-cubes. This class of routing algorithms is based on pipelined circuit-switching. It makes use of the search tree that carries history information. The tree contains full or partial paths starting from the source node containing less than or equal to m misroutes. With the history information distributed throughout the nodes, the probe consists of a vector of dimension offsets, a backtrack flag and a misroute counter. The tree progresses as follows. The routing header starts at the source with all history bits on outgoing arcs cleared. When a routing header leaves a node, the history bit for the arc on which it leaves is set. When a routing header enters a node from above, all descendant history bits are cleared and the ancestor history bit is set. Unless backtracking, a routing header may only leave a node on an arc with a cleared history bit. If and only if all arcs leaving a node have set history bits, the header will backtrack to its immediate ancestor node. When a backtracking header enters a node, history information is not changed. Search terminates when header reaches the destination or backtracks to the root (source node) and all arcs of the source node have history bits set (failure). The maximum path length is $p + 2m$, where p is the minimum path length and m is the number of misroutes.

6.3 Partially Adaptive, Minimal Routing

This section presents the following techniques: safety routing, up preference, virtual channel based *e*-cube routing, multipath *e*-cube routing, multistage dimension-ordered routing, and block-fault model based routing.

Progressive Techniques

Safety Routing

Lee & Hayes [84] described a fault-tolerant communication scheme that facilitates near-optimal routing and broadcasting in hypercube computers subject to node failures. The concept of an ***unsafe*** node is introduced to identify fault-free nodes that may cause communication difficulties in faulty hypercubes. It is then shown that by only using "feasible" paths that try to avoid unsafe nodes, routing and broadcasting can be substantially simplified. A computationally efficient routing algorithm that uses local information is presented which can route a message via a path of length no greater than $p + 2$, where p is the minimum distance from the source to the destination, provided that not all nonfaulty nodes in the hypercube are unsafe. Broadcasting can be achieved under the same fault conditions with only one more time unit than the fault-free case. The scheme is applicable under store-and-forward, virtual cut-through and wormhole routing. The maximum path length is $p + 2$, where p is the minimum distance between the source and destination. The computational complexity of the scheme is $O(n^3)$, where n is the number of hypercube dimensions.

Up Preference

Chiu, Chalsani & Raghavendra [22] proposed a scheme for routing in circuit-switched hypercubes. The scheme is minimal and deadlock-free. It is based on the following concept of ***transitions***. Given a pair of nodes $X = (x_{n-1}, \ldots, x_0)$ and $Y = (y_{n-1}, \ldots, y_0)$, let $UT(X,Y)$ denote the set of dimensions in which X and Y differ such that X has zeros along these dimensions and Y has ones, i.e., $UT(X,Y) = \{i | 0 \leq i \leq n-1, x_i = 0, y_i = 1\}$. $UT(X,Y)$ is referred to as the set of ***up-transitions*** from X to Y. Similarly, $DT(X,Y) = \{i | 0 \leq i \leq n-1, x_i = 1, y_i = 0\}$. $DT(X,Y)$ is referred to as the set of ***down-transitions*** from X to Y. The UP Preference routing criteria is defined such that it allows complete freedom in reserving the up-links and restricts the order in which down-links can be acquired. The algorithm specifies that any up-transition can be made at any point in the order of required transitions, but a down-transition can be made only after all lower dimension transitions have been completed. Computational complexity of the scheme is $O(\sqrt{2^n})$.

Virtual Channel Based E-cube Routing

Sengupta & Bandyopadhyay [122] presented a deadlock-free algorithm for fault-tolerant message routing in wormhole-routed k-ary n-cubes. The e-cube based routing algorithm proposed in [30] is extended with fault-tolerance to design the present algorithm. The scheme uses a two-phase routing. Depending on the set of faulty processors and the n node-disjoint paths that can be used to send messages from a node v_i to v_j. The sender node creates two destinations. One is the actual destination v_j and the other, which is referred to as ***pseudo-destination***, is some v_m in the path selected by v_i to send the message. When v_i sends the message using

one of its outgoing channels, phase 1 of the routing algorithm begins. The message is marked as if it is destined for the pseudo-destination v_m. When the message reaches v_m, phase 1 of the routing is completed and phase 2 begins. The node v_m removes the pseudo-destination information from the flit of the message and forwards it so as to reach the actual destination v_j. The flit of the message contains in addition to the two destination addresses a one-bit flag field to identify the two phases. The algorithm guarantees to find a path of length $O(kn)$ between the source and destination.

Multipath *E*-cube Routing

Draper & Ghosh [36] introduced and evaluated the ***multipath e-cube*** algorithm for adaptive and fault-tolerant wormhole routing in k-ary n-cubes. This algorithm provides multiple shortest paths between any source and destination node of a multicomputer using only a constant number of virtual channels for every physical channel. It also allows multiple broadcast trees to be used concurrently without incurring deadlock.

The proposed algorithm extends the well-known e-cube routing to k-ary n-cubes and also incorporates adaptive routing. In the e-cube routing the dimensions are strictly ordered and are inspected from highest to lowest in order and a link is traversed if and only if the destination address differs from the current address in that dimension. The key idea in the proposed approach is that each dimension should be present more than once in the sequence of dimensions that are inspected for traversal. In its simplest form, the effective dimension order which message traversals follow is $d_{n-1}, d_{n-2}, \ldots, d_0, d_{n-1}, d_{n-2}, \ldots, d_0$, so that each dimension can

be visited twice in the message traversal. Since dimensions are repeated in the dimension ordering of the suggested technique, interdimension cycles occur in the channel dependency graph. Therefore, virtual channels are used to break these cycles.

Multistage Dimension-Ordered Routing

Suh *et. al* [128] presented an approach to fault-tolerant routing in oblivious, wormhole routed networks. The approach is targeted towards environments where the fault rates are relatively low, i.e., on the order of a maximum of 3 failed components between repair cycles.

The basic idea is that when a message encounters a faulty link, it is removed from the network by the local node, which either modifies the header so the message may follow an alternative dimension order path, or computes an intermediate node address. In either case, the message is re-injected into the network. In the case that the message is transmitted to the intermediate node, it will be forwarded upon receipt to the final destination. A message may encounter multiple faults and pass through multiple intermediate nodes. Routing is thus oblivious and is based on wormhole.

The interconnection networks considered are k-ary n-cubes. Adjacent faulty links and faulty nodes are collected into fault regions. Fault regions may overlap forming a larger fault region. Moreover, the fault regions must not disconnect the network and they could be convex or L-shaped or U-shaped concave regions.

Block-Fault Model Based Routing

In the ***block-fault*** model, each fault belongs to exactly one fault block. A set

F of faulty nodes and links indicates a (rectangular) fault block, or f-region, if there is a rectangle connecting various nodes of the mesh such that (a) the boundary of the rectangle has only fault-free nodes or links, and (b) the interior of the rectangle contains all and only the components given by F.

Under the block-fault model, the complete set of faults in a 2D mesh is the union of multiple fault blocks. Conceptually, fault regions may be considered as islands of faults in a sea of communication links and nodes. For each f-region in a network with faults, it is feasible to connect the fault-free components around the region to form a ring or chain. The ***fault ring***, or f-ring of a given region consists of the fault-free nodes and links that are adjacent (row-wise, column-wise, or diagonally) to one or more components of the fault region. Forming a fault-ring around an f-region is not possible when the f-region touches one or more boundaries of the network, in which case, a ***fault-chain***, f-chain, rather than an f-ring is formed around the f-region [9]. A fault-tolerant variant of the *e*-cube algorithm has been developed by Boppana & Chalasani [10], called the ***fcube2*** or ***two-channel fcube*** algorithm, which uses two virtual channels to provide fault-tolerant deadlock-free routing with *e*-cube algorithm as the base. ***fcube2*** may however be applied only to cases in which non-overlapping f-rings exist in the network. For cases in which overlapping f-rings arise a more complicated algorithm called ***fcube4*** has also been suggested which is based on ***fcube2*** and uses four virtual channels in all. The operation of ***fcube2*** is described below.

Let M denote a message in the network. If the current host of M is its destination, then M is consumed. Otherwise, the next hop for the message is determined.

For this purpose, a message status parameter is used which can be set to ***normal*** or ***misrouted***. The status is set as follows. If the next e-cube hop (the path specified by e-cube from the current host to the destination of the message is called its e-cube path; the first hop in that path is its e-cube hop from the current host node) is not blocked by a fault, then set the status of M to ***normal*** and the direction of M to ***null***. Otherwise, set the status of M to ***misrouted*** and set its direction along the f-ring. Once a message's direction is set for the current f-ring, it stays the same throughout its journey around the f-ring. The direction of a misrouted message is reset to null when it becomes a normal message. The status parameter of a message indicates whether the message is blocked by a fault and needs to travel on the corresponding f-ring to reach its destination.

The technique has been extended to two-dimensional and multi-dimensional tori in [15].

6.4 Partially Adaptive, Non-Minimal Routing

This section presents the following techniques: staged routing, safety level based routing, planar-adaptive routing, turn model, and virtual channel based turn model,

Progressive Techniques

Staged Routing

Kim & Shin [68] propose a fault-tolerant routing algorithm that can provide a deadlock-free path between any pair of nodes as long as there exists a healthy Q_{n-2} in an injured Q_n. Q_{n-2}'s are formed by partitioning a Q_n along two dynamically

chosen highest dimensions. When there is no fault in the system, the e-cube routing is used to utilize physical links fully. The scheme classifies all fault situations into 4 distinct cases when an injured Q_n contains at least one healthy Q_{n-2} in which case wormhole routing is used. Each case uses a different routing scheme. Deadlocks are avoided at the expense of the possibility that some of the paths chosen are not always the shortest. Routing decisions are made in a fully decentralized manner, that is, each node makes routing decisions independently of others based only on its own information of faulty components. A ***staged routing*** scheme is proposed for deadlock-free routing in case no healthy Q_{n-2}'s exist in an injured Q_n. In staged routing, messages are stored and forwarded to the destination if the messages use a lower numbered channel than the one previously used.

The wormhole routing algorithm proposed for an injured Q_n that contains at least one healthy Q_{n-2} is called ***multistate detour*** routing which tolerates any number of faults as long as there exists at least one healthy Q_{n-2} in an injured Q_n.

Depending on the number of disjoint healthy Q_{n-2}'s and their locations, all possible fault situations are classified into 4 distinct cases. The method used to establish a path from the source to the destination is different for each case. For the purpose of establishing paths, nodes are grouped according to their locations relative to the failed components. Three of the 4 cases lead to 4 groupings of subcubes and the remaining one case to two groupings. The 4 cases are:

(1) The smallest subcube containing all faulty components of an injured Q_n is a Q_m where $m < n-2$. If A is the Q_{n-2} containing Q_m, then the remaining (healthy) Q_{n-2}'s are B, C, and D.

(2) $m = n - 1$, i.e., the smallest subcube containing all faulty nodes is a Q_{n-1}. The other Q_{n-1} is healthy.

(3) There are two disjoint healthy Q_{n-2}'s.

(4) We can find only one healthy Q_{n-2} from an injured Q_n.

The system is considered non-operational under wormhole routing if there does not exist any healthy Q_{n-2} in an injured Q_n. Each of the above cases is called a system ***state*** which the nodes determine by the fault information broadcast to them. A message in any of the above cases follows the *e*-cube routing or a ***stepwise e-cube*** routing (used in the 3rd case), unless it encounters a fault, in which case it is allowed to take an extra hop detour to a healthy node from where it again follows an *e*-cube path to the destination. In stepwise *e*-cube, the process of finding a path is divided in two phases. The first phase is for the nodes in the subcube to which the source belongs. A node has 3 choices in selecting the next inter-cube node. First, the source node can select a node in one of its adjacent subcubes, with a direct link to it. This new node selects a node in the subcube to which the destination node belongs. If neither of these choices is possible, then the source selects a node in the *e*-cube routing path inside the same subcube which has a connection to one of the adjacent subcubes. All the intermediate nodes try to route messages in the same way as the source. The second phase is for nodes in the subcube to which the destination node belongs. In this phase, *e*-cube routing is used to route messages to the destination after selecting a node in the subcube to which the destination belongs. The computational complexity of the technique is $O(n^2m)$, where m is the number of faulty components.

Safety Level Based Routing

Wu [135] proposed a distributed routing algorithm for hypercubes with faulty nodes using the safety level concept. The ***safety level*** of each node in an n-dimensional hypercube is an approximated measure of the number and distribution of faulty nodes in the neighborhood. Safety level is a concise representation of the distribution of faulty nodes in the system as well. It is also considered as a special case of ***limited global information***, a compromise between local-information and global-information based approaches. Basically, each node in an n-cube is assigned a safety level k, where $0 \leq k \leq n$, and this node is called k-safe. A k-safe node indicates that there exists at least one Hamming distance path (i.e., the optimal path) from this node to any node within k distance. Optimal routing between two nodes is guaranteed if the safety level of the source node is no less than the Hamming distance between the source and the destination. An optimal path is generated by selecting a preferred neighbor with the highest safety level at each routing step. One of the two algorithms presented is an optimal algorithm in which a message is guaranteed to be forwarded to the destination node along an optimal path. The other is a suboptimal algorithm in which a message is forwarded to the destination node along a path with a length of the Hamming distance between the source and the destination plus 2. The selection between the optimal and the suboptimal algorithm can be decided locally at the source node, based on the Hamming distance of the source and the destination, the safety level of the source node, and the safety levels of the neighboring nodes of the source node. At an intermediate node also, a preferred neighbor with the highest safety level is selected for both optimal and suboptimal

algorithms. Maximum path length is $p+2$, where p is the Hamming distance between the source and destination.

Planar-Adaptive Routing

Planar-Adaptive Routing represents a family of routing algorithms, developed by Chien & Kim [20]. These algorithms require only a constant number of virtual channels, independent of the network size and dimension. They reduce the complexity of deadlock prevention by reducing the number of choices at each routing step. In case of network faults, they can be extended with misrouting to produce a working network which remains provably deadlock-free and is provably livelock-free. Planar-adaptive networks tolerate faults by routing around them. The basic idea is to use the flexibility of the adaptive routing algorithm to circumvent any faulty channel. Faulty regions are augmented until they are convex (geometrically, a ***convex*** two-dimensional region is a polygon with no internal angle being larger than 180 degrees). If faulty regions are not naturally convex, healthy nodes and channels are marked as faulty until the regions become convex. Subject to the convexity constraint, a planar-adaptive algorithm routes packets to all parts of the machine which remain connected. Requiring faulty regions to be convex allows a larger fraction of the nodes to remain in service for a given pattern of faults.

The idea in planar-adaptive routing is to provide adaptivity in two dimensions at all times. Thus, a packet is routed adaptively in a series of two-dimensional planes. As the packet progresses towards its destination the routing dimensions change. Eventually, the packet is routed in all dimensions and delivered to its destination. There are two levels of planar-adaptive routing. In ***high-level*** routing, the basic

idea is to route successively in the adaptive planes. After routing in all of the adaptive planes, the packet has reached its destination. In ***low-level*** routing, the scheme is adaptive, as multiple paths can be chosen within each adaptive plane. In each adaptive plane, the packet completes its routing in at least one dimension.

Turn Model

Glass & Ni developed partially adaptive routing algorithms for 2D and 3D meshes [49], [50], [51] and later extended their scheme to hypercubes and k-ary n-cubes [52] augmenting it with fault-tolerance. The algorithms in this scheme do not require addition of physical or virtual channels. First an investigation is made on the possible deadlock cycles in 2D and 3D meshes, then some turns of these cycles are prohibited to prevent deadlocks.

The turn model provides a systematic approach to the development of maximally adaptive routing minimal and non-minimal algorithms, for a given network without adding channels. The following six steps can be used to develop maximally adaptive routing algorithms for nD meshes and k-ary n-cubes.

(1) Classify channels according to the direction in which they route packets.

(2) Identify the turns that occur between one direction and another.

(3) Identify the simple cycles that these turns can form.

(4) Prohibit one turn in each cycle.

(5) In the case of k-ary n-cubes, incorporate as many turns as possible that involve wraparound channels.

(6) Add 180° and 0° turns, which are needed for nonminimal routing algorithms or if there are multiple channels in the same direction.

The fundamental concept behind the turn model is to prohibit the smallest number of turns such that cycles are prevented. Under this model a number of routing algorithms are proposed for 2D meshes which are applicable to higher dimensions also with increasing number of directions. Further discussion of the model and some examples may be found in the section on Fault-Tolerant Routing.

The various proposed algorithms include the ***west-first routing***, ***north-last routing***, ***negative first routing***, and the ***maximally adaptive double-y routing*** or ***mad-y*** for short, all of which are deadlock-free.

The ***west-first routing*** suggests the following: route a packet first west, if necessary, and then adaptively south, east, and north. The ***north-last routing*** suggests the following: route a packet first adaptively west, south and east, and then north. The ***negative-first routing*** suggests the following: route a packet first adaptively west and south, and then adaptively east, and north.

The turn model was also used to improve on the ***double-y*** scheme proposed by Linder & Harden [94], which is a fully adaptive minimal routing scheme. This routing scheme requires two virtual channels in each Y direction (positive and negative) and one virtual channel in each X direction. This set of virtual channels has a total of sixteen 90° turns and four 0° turns. ***Double-y*** allows only eight of the sixteen 90° turns and prohibits all 0° turns. Glass & Ni [51] showed that ***double-y***, although fully adaptive, imposes unnecessary restrictions on routing. They proposed the ***maximally fully adaptive*** (***may-y***) routing algorithm, which makes better use of

the virtual channels and hence improves adaptiveness. It was also shown that a fully adaptive routing algorithm with fewer virtual channels or fewer routing restrictions is not possible without allowing cycles in the channel dependency graph. Thus, it was argued that ***mad-y*** is the least restrictive fully adaptive routing algorithm for 2D meshes.

By applying the turn model to the hypercube, an adaptive routing algorithm, namely p-cube routing has been developed [52]. Let E be the set of all dimension numbers in which the source and destination differ. So a dimension i belongs to E if the ith bits of the source and destination address are different from each other. E_0 is the subset of E consisting of those dimension numbers i, for which the ith bit of the source is 0 and that of the destination is 1. $E_1 = E - E_0$. The fundamental concept of p-cube routing is to divide the routing selection in two phases. In the first phase, a packet is routed through the dimensions in E_0, in any order. In the second phase, the packet is routed through the dimensions in E_1.

Virtual Channel Based Turn Model

Duato [37] proposed a deadlock-free fault-tolerant routing algorithm for wormhole routed n-dimensional meshes. This algorithm is based on the turn model of Glass & and Ni [52]. It splits each physical channel into virtual channels. Only four virtual channels per physical channel are required to avoid deadlock and tolerate faults. The additional virtual channels, if available, are used to increase performance.

Virtual channels are grouped into five sets. Two basic sets of virtual networks provide deadlock-free nonminimal fully adaptive routing. The routing algorithms

for those networks are ***west-last*** (route a packet first adaptively north, south and east, and then west) and ***east-last*** (route a packet first adaptively north, south and west, and then east), respectively, to prevent deadlocks involving the X direction channels. 180° turns are allowed except in the X direction channels. Dimension-ordered routing is used in east (west) border to prevent deadlocks not involving X direction channels. Two extra virtual networks with the same routing algorithm as the basic networks are used to tolerate faults occurring in the east and west borders. Messages are transferred to the corresponding extra virtual network if they find a fault in the east (west) border. The fifth channel set is a pool of channels that can be used by all the messages. Provides $n-1$ alternate paths.

What has been presented above can be summarized in a diagrammatical form as shown in Figures 6.1, 6.2, 6.3, and 6.4. The trees in these figures have been organized based on topology and follow the classification proposed in Chapter 3. It can be seen that most of the work done so has concentrated on the hypercube topology. However, it is also obvious that there is only scarce amount of literature (actually only one) on backtracking based schemes. However, we do have quite a number of fault-tolerant routing techniques for the k-ary n-cube and mesh topologies. Still, backtracking has been overlooked and there are very few techniques that are based on backtracking.

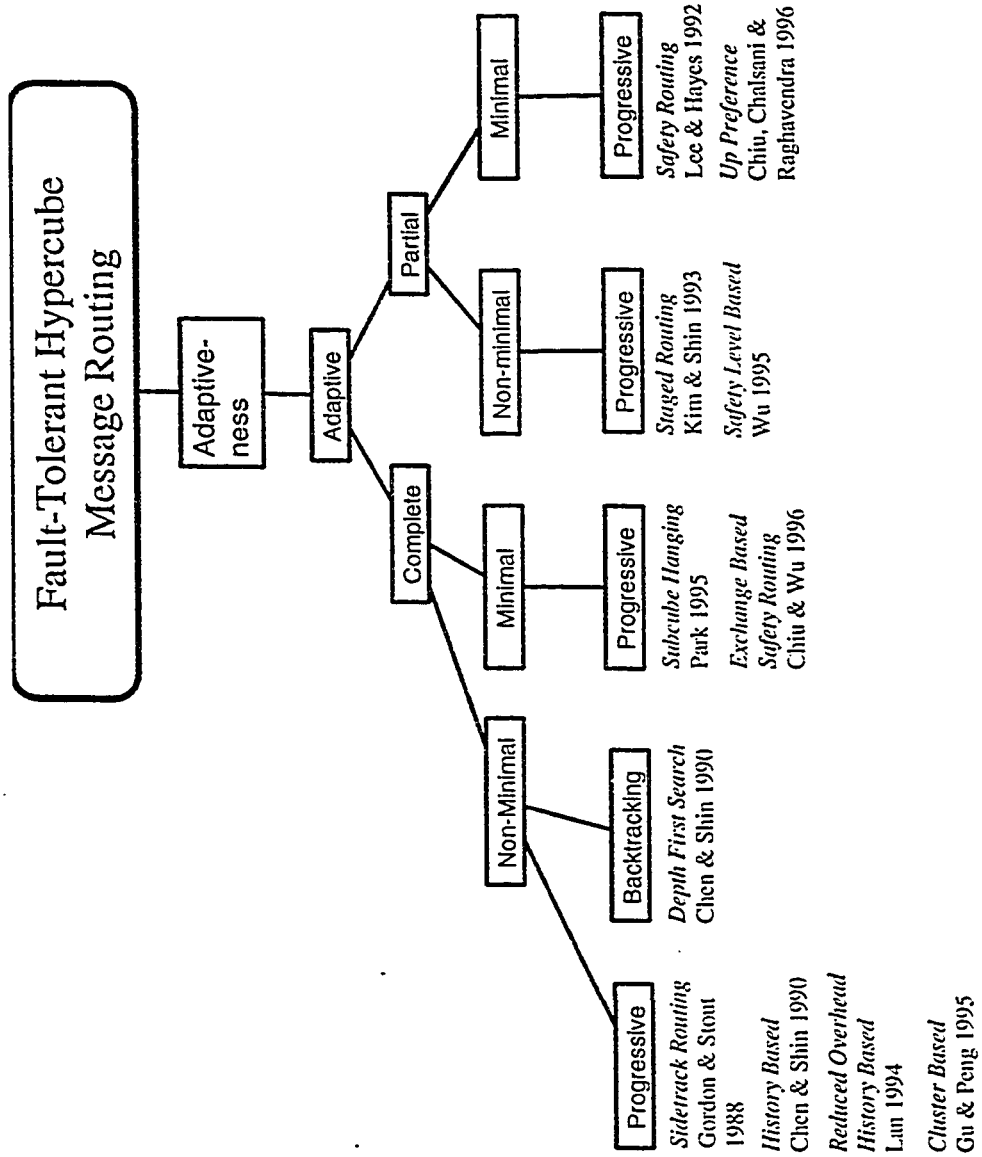


Figure 6.1: Classification of fault-tolerant hypercube routing techniques.

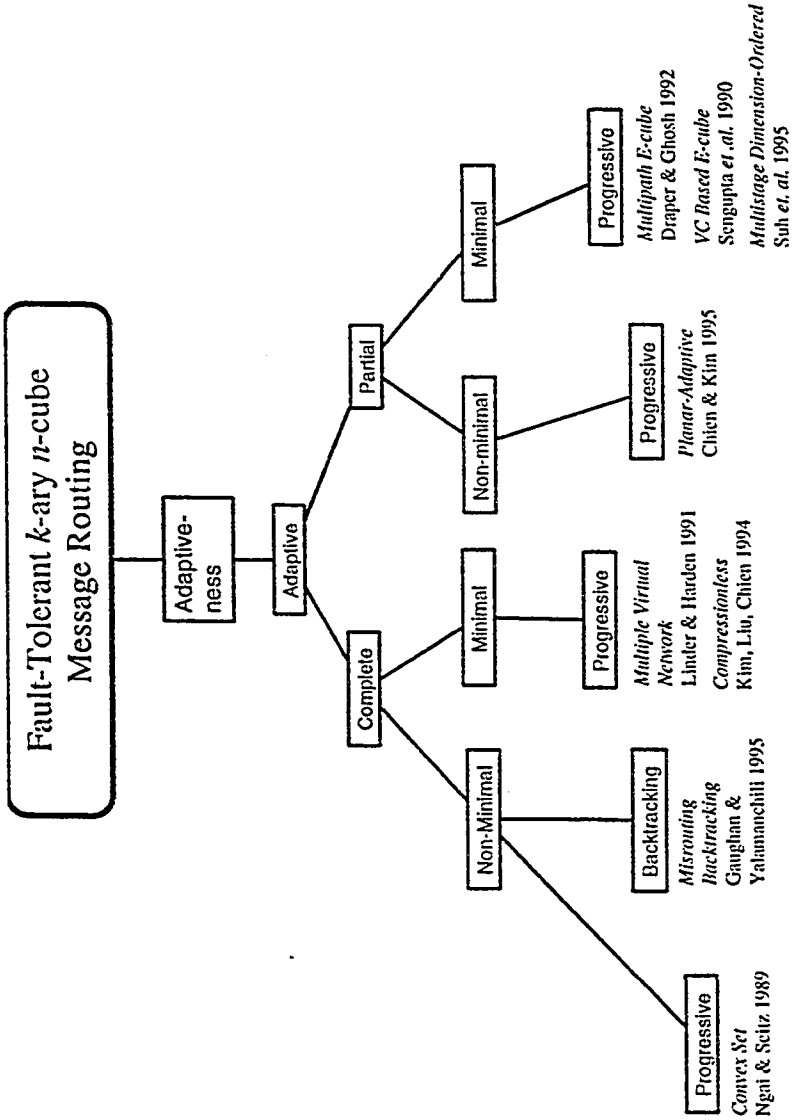


Figure 6.2: Classification of fault-tolerant k-ary n-cube routing techniques.

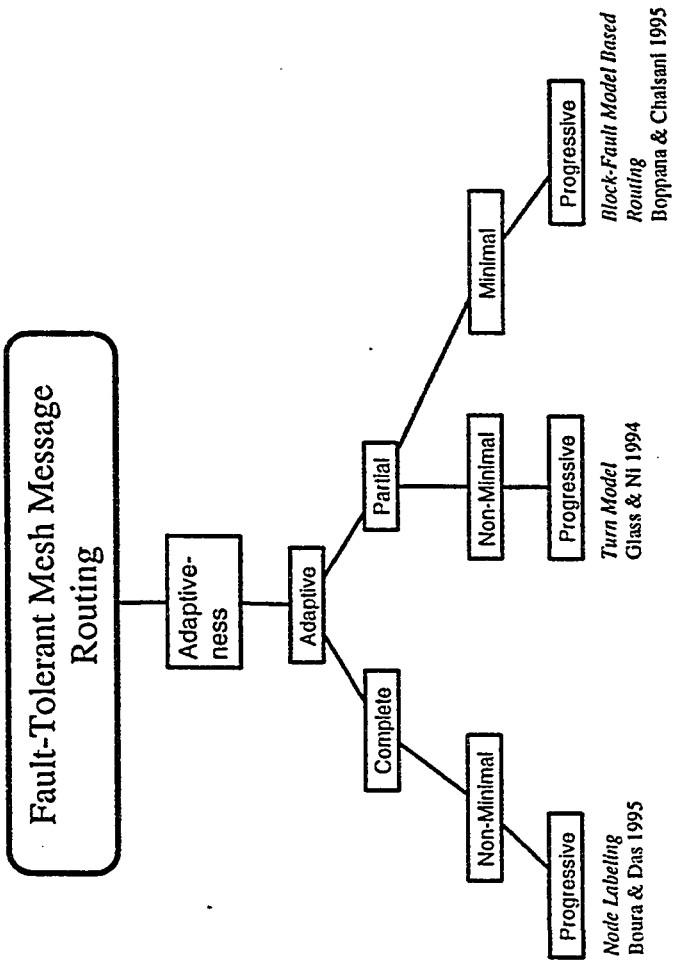


Figure 6.3: Classification of fault-tolerant mesh routing techniques.

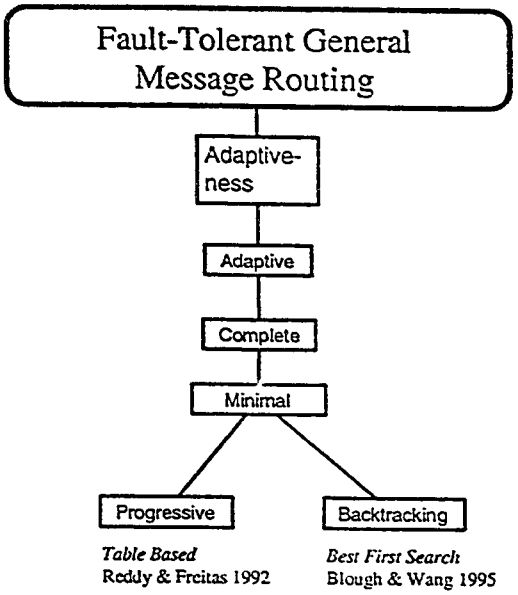


Figure 6.4: Classification of fault-tolerant general routing techniques.

6.5 Comparison of Fault-Tolerant Routing Techniques

A comparative study of the fault-tolerant techniques for hypercube networks is presented in Tables 6.1 and 6.2. Table 6.1 compares the techniques in terms of the switching mechanism used, various adaptiveness characteristics of the technique as proposed in the classification, and the number of virtual channels used. Table 6.2 compares them in terms of the fault information model, the fault model, and the number of faults tolerated. All the techniques are deadlock- and livelock-free.

Sidetrack routing [53] may cause deadlock and livelock. *History based* routing [16, 18] guarantees the maximum path length to be $O(log_2\ n)$. *Reduced overhead history based* technique [78] possesses the same characteristics, except that the his-

tory information kept in the routing header is reduced, thus making the scheme efficient. The *cluster based* routing strategy [56] guarantees the maximum path length to be at most $n+2$. *Subcube based hanging* [110] guarantees the maximum path length to be at most $\lfloor \frac{n-1}{log_2(n-1)} \rfloor + p$, where p is the Hamming distance between the source and destination. *Exchange based* routing [21] guarantees the maximum path length to be at most $p+4$, where p is the hamming distance between the source and the destination.

Table 6.1: Comparison of Fault-Tolerant Hypercube Routing Techniques

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Side-tracking	SF	Complete	Non-Minimal	Prog.	NA
Safety Routing	SF, CS VCT, WH	Partial	Minimal	Prog.	NA
History Based	SF	Complete	Non-Minimal	Prog.	NA
Red Ovhd His. Based	SF	Complete	Non-Minimal	Prog.	NA
Depth-First Search	SF	Complete	Non-Minimal	Back.	NA
Up-Pref.	CS	Partial	Minimal	Prog.	NA
Cluster Based	SF, CS VCT, WH	Complete	Non-Minimal	Prog.	NA
Safety Level	SF, CS VCT, WH	Partial	Non-Minimal	Prog.	NA
Subcube Based Hang.	WH	Complete	Minimal	Prog.	See Sec. 6.1
Exchange Based	SF, CS VCT, WH	Complete	Minimal	Prog.	5 VC/PC
Staged Routing	WH	Partial	Non-Minimal	Prog. Back.	NA

Table 6.2: Fault-Tolerant Properties of Hypercube Routing Techniques

Technique	Fault Info	Fault Model	No. of Faults
Side-tracking	Local	Node Link	Any as long as network is conn.
Safety Routing	Local	Node	Any as long as network is conn.
History Based	Local	Node Link	$< n$
Red. Ovrhd. His. Based	Local	Node Link	$< n$
Depth-First Search	Local	Node Link	Any as long as network is conn.
Up-Pref.	Local	Node Link	$m < n$, m is size of min. subcube containing all faults
Cluster Based	Local	Node	$\leq 2n - 3$
Safety Level	Limited Global	Node	Any as long as network is conn.
Subcube Based Hang.	Local	Node	$< n$
Exchange Based	Local	Node	$< n$
Staged Routing	Local	Node Link	$< 2n$

A comparative study of the fault-tolerant techniques for k-ary n-cube networks is presented in Tables 6.3 and 6.4. Table 6.3 compares the techniques in terms of the switching mechanism used, various adaptiveness characteristics of the technique as proposed in the classification, and the number of virtual channels used. Table 6.4 compares them in terms of the fault information model, the fault model, and the number of faults tolerated. All the techniques are deadlock- and livelock-free.

Table 6.3: Comparison of Fault-Tolerant k-ary n-cube Routing Techniques

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Convex Set	Adap. VCT	Complete	Non-Minimal	Prog.	NA
Multipath E-cube	WH	Partial	Minimal	Prog.	4 VC/PC
Multiple Vir. Net.	WH	Complete	Minimal	Prog.	$(n+1)2^{n-1}$ VC/PC
Compress-ionless	WH	Complete	Minimal	Prog.	NA
Planar Adaptive	WH	Partial	Non-Minimal	Prog.	3 VC/PC
VC Based E-cube	WH	Partial	Minimal	Prog.	NA
Multistage Dim. Ordered	WH	Partial	Minimal	Prog.	NA
Misrouting Backtracking	PCS	Complete	Non-Minimal	Back.	Any no. of VCs necessary

A comparative study of the fault-tolerant techniques for mesh networks is presented in Table 6.5 and 6.6. Table 6.5 compares the techniques in terms of the switching mechanism used, various adaptiveness characteristics of the technique as proposed in the classification, and the number of virtual channels used. Table 6.6 compares them in terms of the fault information model, the fault model, and the

Table 6.4: Fault-Tolerant Properties of k-ary n-cube Routing Techniques

Technique	Fault Info	Fault Model	No. of Faults
Convex Set	Local	Node	Any as long as network is conn.
Multipath E-cube	Local	Link	n
Multiple Vir. Net.	Local	Node	1
Compress-ionless	Local	Node Link	Any as long as network is conn.
Planar Adaptive	Local	Block	Any as long as network is conn.
VC Based E-cube	Local	Node	$< n$
Multistage Dim. Ordered	Local	Block	Any as long as network is conn.
Misrouting Backtracking	Local	Node Link	$2n - 1$

number of faults tolerated. All the techniques are deadlock- and livelock-free.

A comparative study of the fault-tolerant techniques for general networks is presented in Table 6.7 and 6.8. Table 6.7 compares the techniques in terms of the switching mechanism used, various adaptiveness characteristics of the technique as proposed in the classification, and the number of virtual channels used. Table 6.8 compares them in terms of the fault information model, the fault model, and the number of faults tolerated. All the techniques are deadlock- and livelock-free.

Table 6.5: Comparison of Fault-Tolerant Mesh Routing Techniques

Technique	Switch. Tech.	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Node Labeling	WH	Complete	Non-Minimal	Prog.	3 VC/PC
Turn Model	WH	Partial	Non-Minimal	Prog.	NA
Block Based	WH	Partial	Minimal	Prog.	4 VC/PC
VC Based Turn Model	WH	Partial	Non-Minimal	Prog.	4 VC/PC

Table 6.6: Fault-Tolerant Properties of Mesh Routing Techniques

Technique	Fault Info	Fault Model	No. of Faults
Node Labeling	Local	Node	Any as long as network is conn.
Turn Model	Local	Link	$< n$
Block Based	Local	Block	Any as long as network is conn.
VC Based Turn Model	Local	Node	$< n - 1$

Table 6.7: Comparison of Fault-Tolerant General Techniques

Technique	Switch. Tech.	Topology	Complete/ Partial	Minimal/ Non-Minimal	Prog./ Back.	VCs
Table Based	SF, CS VCT, WH	Any Topology	Complete	Minimal	Prog.	NA
Best-First Search	SF	Any Topology	Complete	Minimal	Back.	NA

Table 6.8: Fault-Tolerant Properties of General Techniques

Technique	Fault Info	Fault Model	No. of Faults
Table Based	Global	Node Link	Any as long as network is conn.
Best-First Search	Limited Global	Node	Any as long as network is conn.

6.6 Summary

In this chapter we have discussed the various fault-tolerant routing techniques. Different characteristics of each technique have been given. Also, a brief discussion on the operation of each technique is presented. The techniques have been organized according to the classification presents earlier. Moreover, a comparison has been made between the various fault-tolerant routing techniques. The comparison has been based on the classification presented earlier together with the topology and other features such as the complexity, the size of the routing path, etc.

Chapter 7

Routing Techniques for Broadcasting and Multicasting

In order to support the operation of massively parallel computer systems, there is a need to provide fast communication mechanisms that may be used to distribute the load among the various processors or for the processors to communicate the results of computations. There are two types of communication operations, one in which a single node is allowed to communicate a message to only a single destination which may be its immediate neighbors or others, called *unicast* or *one-to-one* or *point-to-point*; the second is a large class of various operations whereby either multiple sources or multiple destinations or both are permitted, called *collective communication*. The most important operations defined in this set are *broadcast* and *multicast*. In *broadcast*, one node sends the same message to all other group members. *Multicast* refers to a type of communication operation in which one source wants to send its messages to k distinct destinations.

Broadcast is a very common operation needed in a parallel system to distribute code and data from a host processor to a set of node processors before computation begins. This operation is also used to distribute data from one processor to others during the computational phase of a distributed memory program. In any parallel system, a broadcast operation results in a multicast operation if not all processors participate in the operation.

We have already presented routing techniques for one-to-one routing in Chapters 5 and 6. In this chapter a study is presented of the various broadcast and multicast routing techniques. These are organized by classifying the techniques into either broadcast or multicast routing techniques. The chapter is thus organized into two subsections, first consisting of the techniques for broadcast routing and the second consisting of the techniques for multicast routing.

7.1 Broadcast Routing

This section consists of the description of following techniques: coordinated recursive doubling, safety based broadcast, multipath e-cube, message partitioning, double tree, dimension broadcast tree, and dimension simple path scheme.

7.1.1 Fault-Tolerant Techniques

Coordinated Recursive Doubling

Ramanathan & Shin [113] proposed a broadcast algorithm for hypercube multicomputers containing faulty nodes/links. The algorithm delivers multiple copies of the

broadcast message through disjoint paths to all the nodes in the system. The assumption is that no information on faults is available to any node. The basic idea of the algorithm is as follows. The node that wants to broadcast a message sends the message to all its neighbors. The neighbors in turn broadcast the message they received using a simple coordinated recursive doubling algorithm. The recursive doubling algorithm executed by the neighbors is coordinated such that the copies of the message received by a node have traveled through disjoint paths. The algorithm basically broadcasts a message n times in a way that enables the message to reach a nonsource node via n node-disjoint paths. A node can therefore, compare the copies of the messages it receives and choose one by majority. The number of time units required to broadcast a message is $n+1$ if a node can send multiple broadcast messages at a time, or $2n$ if a node can send only a single message per unit time. This scheme tolerates $n-1$ node failures at the cost of generating a large amount of message traffic if faulty nodes do not forward messages. Only $\lfloor n/2 \rfloor$ faults are tolerated if faulty nodes forward corrupted messages.

Safety Based Broadcasting

Lee and Hayes [84] describe a fault-tolerant communication scheme that facilitates near-optimal broadcasting in hypercubes subject to node failure. The concept of an ***unsafe node*** is introduced to identify fault-free nodes that may cause communication difficulties. In order to broadcast messages from an active node to all nonfaulty nodes in n time units, the algorithm proceeds as follows. Suppose that each nonfaulty node has unsafe and faulty node lists which cover all nodes in its 1-neighborhood. An n-bit control word is sent along with each message. This is

used to record each link not traversed because the nodes connected by it are unsafe or faulty, and to provide information telling the receiving node how to continue broadcasting. Initially all bits in the control word are set to 1. An active node x sends a message to an active neighbor via link i if the corresponding bit i is 1; it sends a message to an unsafe neighbor only if bit i of the control word is 1 and all other bits are 0. This bit i is set to 0 when the message is sent out by x. If x has to broadcast a message to several nearest neighbors, one of which is unsafe, the unsafe neighbor is the last to receive the message. With this arrangement, no unsafe nodes have to transmit messages since they receive the all-0 control word. The computational complexity of the scheme is $O(n^3)$ and the number of message passing steps to complete the broadcast is $n + 1$.

Multipath E-cube Broadcasting

Draper & Ghost [36] proposed a ***multipath e-cube algorithm*** for e-cube based adaptive fault-tolerant wormhole broadcasting in k-ary n-cubes. It allows multiple broadcast trees to be used concurrently without incurring deadlocks. It generalizes the concept of SBT (spanning binomial tree) to obtain a scheme for constructing broadcast trees in k-ary n-cubes. The broadcast scheme is defined as follows. The source node sends the broadcast message over all its outgoing channels. A receiving node takes the following action upon reception of the message. If a message is traversing a dimension in the positive direction, it is forwarded in the same positive dimension. If the incoming message is traversing a dimension in the negative direction, it is forwarded in the same negative direction. Regardless of the direction of the incoming message, any broadcast message received along a channel correspond-

ing to a dimension is forwarded in both directions in all dimensions smaller than the incoming direction.

7.1.2 Non-Fault-Tolerant Techniques

Message Partitioning Approaches

As in EDST broadcast algorithm [62], it is possible to take advantage of the fact that all nodes participate in the broadcast operation and are available to relay and replicate the message. Such approaches involve partitioning the message and sending parts of it along separate subnetworks, with every node eventually collecting all the pieces. One such algorithm that uses this approach in 2D mesh networks is called ***edge-disjoint spanning fences*** (EDSF) [4]. The EDSF algorithm embeds two disjoint spanning trees, or fences, in the mesh. The source node partitions the message and alternately pipelines the pieces through the two fences. To accommodate any node as the source, the mesh is actually considered as a logical torus, with some segments being wormholed across the network to the west and north. Therefore, the same algorithm can be applied to an actual torus as well. Compared to the single tree approaches, this approach involves many more message startups that accompany message partitioning. Despite these additional costs, however, this approach has been shown to provide better performance, for relatively long messages, than a non-pipelined tree.

Another broadcast method that involves message partitioning, called ***scatter-collect***, has also been proposed in [4]. In this technique, the source node partitions the message and performs a scatter operation (an operation in which one node

sends a different message to each of a set of destinations) across its row, with each node receiving a particular segment of the message. Next, each node collects the remaining segments. In the 2D mesh, the collect operation is implemented as two operations: collecting in rows and then in columns. Specifically, the nodes in each row form a logical ring; nodes circulate segments around the ring until all nodes hold all segments in that row. Next, the nodes in each column form a logical ring, and nodes circulate segments around the ring until all nodes hold all segments. This algorithm avoids channel contention and the pipelining of messages provides good performance for broadcast of long messages. The scatter-collect algorithm can be used to implement multicast in hypercubes, meshes and tori, with the aid of the multicast tree to scatter the data to all the nodes in the group. Next, the nodes in the group can form a logical ring, according to a dimension-ordered chain, and circulate the message segments until all destinations have received all segments.

Double Tree Approach

McKinley & Trefftz [99] proposed a simple variation on the SBT (spanning binomial tree) called the ***Double Tree*** (DT) algorithm, which is designed for all-port, wormhole-routed hypercubes. It is an attempt to reduce the broadcast time by taking advantage of the distance insensitivity of wormhole routing and the presence of multiple ports between processors and their routers. The fundamental concept behind the proposed method is to reduce broadcast latency by relaxing the common assumption that all constituent unicast messages must be sent between neighboring nodes. the DT algorithm begins with the source node s sending to node $\bar{s}$, whose address is the bitwise complement of s. Subsequently, each of the nodes s and $\bar{s}$

becomes the roots of partial spanning binomial trees. The tree rooted at s is called the ***forward tree***, and the one rooted at $\bar{s}$ is called the ***backward tree***. With the proposed technique, the number of message passing steps required to reach all nodes in an n-dimensional hypercube is $\lceil n/2 \rceil$. Although the DT algorithm reduces broadcast latency compared to the SBT algorithm, it is not optimal for large hypercubes [97].

Dimension Broadcast Tree Approach

The one-to-all broadcast problem in one-port wormhole-routed mesh networks was first studied by Barnett *et. al.* [4]. The algorithm presented operates in a recursive manner, similar to the spanning binomial tree, within each dimension of the mesh. Suppose that the source node is the leftmost node in a linear array topology. In the first message-passing step, the source node sends the message halfway across the linear array, partitioning the network into two subnetworks. In subsequent steps, each node holding a copy of the message forwards it to a node in its partition that has not yet received the message. By employing non-nearest-neighbor communication, this approach takes advantage of the pipelining effect of wormhole routing.

The algorithm may be generalized to higher-dimensional meshes by applying the above procedure to one dimension at a time. An important feature of this algorithm is that by partitioning the network in this manner, channel contention among the messages is avoided. In general this algorithm requires $\sum_{i=1}^{d} \lceil log\ w_i \rceil$ message-passing steps in a d-dimensional mesh in which the width of dimension i is w_i.

Dimension-Simple Path Scheme

Ho and Kao [59] presented an optimal algorithm that broadcasts on an n-dimensional hypercube, in which the network is recursively divided into subcubes of nearly equal size; the DT approach [99] is used to finish the broadcast operation in small subcubes. The approach uses the concept of a dimension-simple path. A path $P = v_0, v_1, \ldots, v_d$ in an n-dimensional hypercube is called ***dimension-simple*** if there exists a sequence $i_1, i_2, \ldots, i_d$ of distinct cube dimensions such that for all $v_i, j \geq 1$, v_j is obtained from v_{j-1} by complementing the bit at dimension i_j, where v's represent the node addresses and d is the length of the path. The path P is called ***ascending*** (***descending***), if $i_1 < i_2 < \ldots < i_d$ ($i_1 > i_2 > \ldots > i_d$).

In an all-port wormhole-routed hypercube in which dimension-ordered routing is performed by resolving addresses from top-to-bottom, a single node can send a message to all the nodes along an ascending path simultaneously, except for any message staggering due to sending latency. In this manner, an n-dimensional hypercube can be partitioned into $n+1$ subcubes such that each subcube contains one node on the dimension-simple path. It has also been shown that any cube can be partitioned equally or nearly equally, which implies that the number of steps required by the algorithm is optimal to within a multiplicative constant, specifically, the time required is $\Theta(\frac{n}{log_2(n+1)})$.

7.1.3 Comparison of Broadcast Routing Techniques

Now we present a comparison among the various broadcast routing techniques. The comparison is made on the basis of the switching technique used, the topology, the

port model, and fault-tolerance. Additional bases such the time complexity of an algorithm, number of time steps to complete the broadcast, and the maximum path length guaranteed by the technique are also used. In these comparisons we follow the organization of Table 7.1.

The *coordinated recursive doubling* technique [113] requires $n+1$ time units to broadcast a message is $n+1$ if a node can send multiple broadcast messages at a time, or $2n$ if a node can send only a single message per unit time. The *safety based* technique [84] has a computational complexity of $O(n^3)$ and number of message passing steps required to complete the broadcast is $n+1$. The *double tree* technique [99] requires $\lceil n/2 \rceil$ time steps to complete the broadcast. The *dimension simple* technique [59] requires $\Theta(\frac{n}{log_2(n+1)})$ time steps to complete the broadcast. *Dimension broadcast tree* [4] requires $\sum_{i=1}^{d} \lceil log w_i \rceil$ message-passing steps in a d-dimensional mesh in which the width of dimension i is w_i.

7.2 Multicast Routing

This section consists of the description of following techniques: sidewalk multicast, lookahead multicast, safety level multicast, best-fit multicast, greedy multicast, multicast routing conforming to base routing, double channel XY multicast routing, pathlike multicast, column path multicast, virtual channel based multicast, unicast based multicast, and multipacket multicast.

Table 7.1: Comparison of Broadcast Routing Techniques

Technique	Switch. Tech.	Type	Topology	Port Model	Fault Tolerance
Coordinated Recursive	SF	Brdcst	Hypc	1-port	no. faulty nodes $< n$ Local Info.
Safety Based	SF, CS VCT, WH	Brdcst	Hypc	1-port	no. faulty nodes $< \lceil n/2 \rceil$ Lim. Global
Multipath E-cube	WH	Brdcst	k-ary n-cube	n-port	FT, Local Info. $< n$
Message Partitioning	WH	Brdcst	2D-Mesh	1-port	non-FT
Double Tree	WH	Brdcst	Hypc	n-port	Non-FT
Dimension Broadcast Tree	WH	Brdcst	nD-Mesh	1-port	non-FT
Dimension Simple	WH	Brdcst	Hypc	n-port	Non-FT

7.2.1 Fault-Tolerant Techniques

Sidewalk Multicast

Lan [77, 79] proposed a fault-tolerant multicast routing algorithm based on the OMT formulation of [80, 81]. At each forward node, all the destinations in the destination list vote for the most favorable dimensions to pass the message. The destinations are divided into several sublists and send to the selected dimensions correspondingly. However, in faulty hypercubes, faults cause new problems to be dealt with as follows. Each fault-free node keeps the knowledge of all dimensions in a ***prohibit-vector*** $X = x_{n-1}, \ldots x_0$, such that if dimension j is faulty, $x_j = 0$; otherwise $x_j = 1$. A node is called ***semi-isolated*** if it has only one non-faulty

dimension. Each fault-free node keeps a ***semi-isolated vector*** $S = s_{n-1}, \ldots, s_0$, such that if the neighbor at dimension j is semi-isolated, $s_j = 0$; otherwise $s_j = 1$.

Special care is taken to avoid possible looping or livelock. Since faults may have arbitrary distribution and each fault-free node has only local information, it may happen that node u sends a message to a neighboring node v, destined for node z. But the only nonfaulty dimension at node v is the link connecting u and v. In this case, if the message is allowed to pass back to u, then u may send it back to v again. To prevent this situation from happening, the semi-isolated vector is used. If a forward node has a semi-isolated neighbor, it sends messages to that neighbor only when the neighbor is a destination. That neighbor will not be asked to pass message to any other destinations. The path length to each destination is at most $H + 2F$, where H is the Hamming distance between the source and destination and F is the number of faults.

Lookahead Multicast Routing

Liang *et. al.* [87] consider the multicast problem for a d dimensional binary hypercube in the presence of F link failures, where $F < d$. A fault-tolerant multicast technique has been developed called ***lookahead multicast*** which is deadlock-free. It can ensure successful multicast when each node knows the status of all the links within two hops. The lookahead algorithm is executed by the source node and every forward node in the path to the destination. It uses the concept of ***necessary dimensions*** ($ND_{x,y}$), which is defined at an intermediate node q_x for a given destination q_y as the set of dimensions that need to be resolved to reach q_y from q_x. The ***spare dimension*** ($SD_{x,y}$) for a given destination q_y at q_x includes all those

dimensions which are not included in $ND_{x,y}$.

When a node q_x receives a message, it checks k (where k is the Hamming distance of the destination of the message and the node q_x) in the message. If $k = 0$, q_x is the destination of the message; otherwise, the lookahead algorithm scans each of the faulty dimensions. With each faulty dimension j, it checks whether there is any destination that might exist along dimension j. If so, it finds a lookahead path from q_x to the neighbor of q_x along the faulty dimension consisting of nodes along ND links, and sends the corresponding destinations (along with the message copy) along the that lookahead path.

This scheme can tolerate up to $n-1$ faulty links and the number of additional time steps required in the worst case is $2n$ in an n-dimensional hypercube. The maximum path length is $H + 2\lfloor(\sqrt{8F+1} - 1)/2\rfloor$, where H is the Hamming distance between the source and destination and F is the number of faults. The computational complexity of the scheme is $O(log_2\ n)$.

Safety Level Based Multicasting

Wu [134] studied the multicast problem in hypercubes with faulty nodes. A node is assumed to have limited global information of fault distribution, captured by the safety level associated with the node. Basically safety level is an approximate measure of the distribution of faulty nodes in the neighborhood. A safety-level based multicasting scheme which guarantees time optimality has been proposed. Three approaches, safety-level multicasting (SLBM), modified safety-level based multicasting (MSLBM), and address-sum based multicasting (ASBM) were proposed. In SLBM, the priority of a dimension is determined a priori based on the safety level of

the neighbor along this dimension. The higher the safety level of the neighbor along this dimension, the higher the priority order of this dimension. Two approaches can be used when there are two or more dimensions along which the corresponding neighbors have the same highest safety level. In SLBM, a priority order among these dimensions is randomly selected. In MSLBM, the priority of a dimension is based on the corresponding bit value in the address summation of the destinations. Basically if the neighbor along dimension d can carry maximum possible destination nodes, then d has the highest priority. When there is more than one bit that has the same maximum value in the address summation, the selection is random. The next highest priority dimension is determined using the same approach but based on the updated destination set, i.e., the one after removing those nodes to be forwarded to dimensions in higher priorities.

In ASBM , the dimension priority depends primarily on bit values in the address summation. A dimension has the highest priority if the corresponding neighbor can carry the maximum possible nodes. To ensure time optimality, only those destination nodes that are no more than k distance away from the selected neighbor will be included, where k is the safety level of this neighbor. In this case, the safety levels of all the neighbors and the relative distances of the destination nodes are used in the decision. A modified ASBM approach, following a similar approach as used in MSLBM, can be adopted when there is more than one neighbor that can carry the same maximum number of destination nodes. In this case, the priority among these neighbors is based on their safety levels. In ASBM, the priority among these nodes is randomly selected. The maximum path length in the safety level based scheme

is $< H + 2$, where H is the Hamming distance between the source and destination. The computational complexity of the scheme is $O(n)$.

7.2.2 Non-Fault-Tolerant Techniques

Bestfit Multicast

Sager & McMillin [120] presented a bestfit distributed multicast routing algorithm which has $O(k\ log\ n)$ complexity on an n-dimensional hypercube, where k is the number of destination nodes. In this algorithm, the multicast communication travels to each destination along a shortest path. It can be easily implemented in hardware. It does not necessarily avoid deadlock. The algorithm basically tries to place each destination on the channel to which it fits best. That is, the current destination should look "a lot like" the "average" destination already assigned to the chosen channel. A destination is assigned to an empty channel if and only if its assignment to each non-empty channel would violate the constraint that for each channel there is at least one bit position for which all destinations assigned to that channel differ from the address of the local node.

Greedy Multicast

Lan [80, 81] presented a graph theoretical model, the ***Optimal Multicast Tree*** (OMT), for interprocessor communication in multicomputer systems, which can be defined as follows. Let $(G(V, \dot{E})$ be a graph corresponding to the interconnection topology of a multicomputer under consideration and $D = \{u_2, u_2, \ldots, u_k\} \subset V$ denote the set of k destination nodes in a multicast. The set $M = \{u_0\} \cup D$ is

referred to as a ***multicast set***, where node u_0 represents the source node.

The multicast problem of finding a subtree $T(V, E)$ of $G(V, E)$, called the ***Optimal Multicast Tree*** (OMT), is such that:

(a) $M \subseteq V(T)$,

(b) $d_T(u_0, u_1) = d_G(u_0, u_1)$, for $1 \leq i \leq k$, and

(c) $|E(T)|$ is as small as possible.

where $E(T)$ and $V(T)$ are the edge set and vertex set of the tree $T(V, E)$, respectively.

A subtree of G which satisfies conditions (a) and (b) above is referred to as a ***multicast tree***. In a multicast tree, a ***leaf*** node is a destination node of degree one, and all nonleaf nodes are referred to as ***forward*** nodes. Thus, an ***intermediate*** node refers to a nondestination forward node. In general an OMT may not be unique.

Based on the above model a greedy multicast algorithm for hypercubes has been proposed which works as follows. Each node, upon receiving a multicast message, will perform the following functions. First, it will compare its own address against the address in the destination list of the received message. If there is a match, that matched address will be removed from the destination list. Then, if the destination list is empty, the node is a leaf node and no message will be further forwarded. However, if the destination list is not empty, the node will determine its descending neighbors in the multicast tree. To do this, each forward node can use the relative binary addresses of all its destination nodes to ***vote*** for preferred dimensions (bit

positions). Each of the n bit positions has a counter For each destination, if its binary address has value 1 at c bit positions, the corresponding c counters are increased by 1. Then, if the counter of a particular bit position, say j, has the minimum number of 1's, the jth -dimensional neighbor of the forward node u_0 is selected. In case of a tie, select one of the tied bit positions at random.

All the destination nodes whose binary addresses have value 1 at bit position j are selected to form a destination sublist in the message sent to the jth-dimensional neighbor. This implies that the neighbor is responsible for passing the message to all those destinations in the list. The same procedure continues for the remaining destinations which do not have 1 at their bit position j. This procedure is repeated until all destination nodes have been resolved. The computational complexity of the scheme is $O(nk + n^2)$, where k is the number of destinations. The number of time steps required to complete the broadcast is $O(n)$.

Multicast Routing Conforming to Base Routing

Panda *et. al.* [109] have proposed a general technique to provide multicast routing capability using the existing unicast routing method. The unicast routing algorithm can be *e*-cube or an adaptive algorithm. As in the path based algorithms, the set of destinations of a multicast message is partitioned such that each subset of the destinations can be serviced by a single message. Each such message visits its destinations in a specific order such that its entire path is a valid path for a unicast message between the source of the multicast and the last destination of the message If the underlying unicasting routing is *e*-cube, then a message may visit multiple destinations some in row and some in column. Because a multicast message in this

scheme can service destinations in more ways, this tends to reduce the number of messages used. However, this also creates more dependencies on other resources.

Double Channel XY Multicast Routing

Lin *et. al.* [90] proposed a deadlock-free multicast wormhole method based on XY routing which avoids cyclic channel dependencies by doubling each channel in the 2D mesh. The resulting network is partitioned into four subnetworks, $N_{+X,+Y}, N_{-X,+Y}$, $N_{+X,-Y}$, and $N_{-X,-Y}$. Subnetwork $N_{+X,+Y}$ contains the unidirectional channels with addresses $[(i,j),(i+1,j)]$ and $[(i,j),(i,j+1)]$; subnetwork $N_{+X,-Y}$ contains channels with addresses $[(i,j),(i+1,j)]$ and $[(i,j),(i,j-1)]$; and so on.

For a given multicast, the destination node set D is divided into at most four subsets, $D_{+X,+Y}, D_{-X,+Y}, D_{+X,-Y}$, and $D_{-X,-Y}$, according to the relative positions of the destination nodes and the source node u_0. Set $D_{+X,+Y}$ contains the destination nodes to the upper right of u_0, $D_{-X,+Y}$ contains the destinations to the upper left of u_0, and so on. The multicast is thus partitioned into at most four independent submulticasts from u_0 to each of $D_{+X,+Y}, D_{-X,+Y}, D_{+X,-Y}$, and $D_{-X,-Y}$. The submulticast to $D_{+X,+Y}$ will be implemented and the message will be sent to the destination nodes through subnetwork $N_{+X,+Y}$ using XY routing, $D_{-X,+Y}$ in subnetwork $N_{-X,+Y}$, and so on.

Pathlike Multicast Routing

Lin *et. al.* [90] proposed a network partitioning strategy based on Hamiltonian paths. A ***Hamiltonian path*** visits every node in a graph exactly once. In a network with N nodes, the assignment of the label to a node is based on the position of that

node in a Hamiltonian path, where the first node in the path is labeled 0 and the last node in the path is labeled $N-1$. The algorithm effectively labels the network dividing it into two subnetworks. The ***high-channel subnetwork*** contains all of the channels whose direction is from lower-labeled nodes to higher-labeled nodes, and the ***low-channel subnetwork*** contains all of the channels whose direction is from higher-labeled nodes to lower labeled nodes. Unicast and multicast communication use the labeling for routing. That is, a unicast message follows a path based on labeling instead of XY routing. If the label of the destination node is greater than the label of the source node, the routing always takes place in the high-channel network; otherwise, it will take place in the low-channel network.

Column-Path Multicast Routing

Boppana, Chalsani & Raghavendra [12] proposed a multicast routing algorithm for mesh-based wormhole-switched multicomputers called ***column-path*** and its fault-tolerant variant [11] which can tolerate multiple convex faults. It uses the *e*-cube algorithm for multicast routing. This algorithm partitions the set of destinations of a multicast message into at most $2k$ subsets (k is the number of columns in the mesh), such that there are at most 2 messages directed to each column. Only one message is sent to a column if all destinations in that column are either below or above the source node; otherwise, two messages are sent to that column. The column-path algorithm is compatible with the unicast routing methods used in the current parallel computers.

Virtual Channel Based Multicasting

Duato [41] developed a theory for the design of deadlock-free wormhole-routed adaptive multicast routing. Using this theory, a multicast routing algorithm is designed for 2D meshes based on virtual channels which is an extension of the algorithm proposed in [90]. The algorithm has two parts, the first part splits the destination set for a message into two or more subsets, then it sorts each subset. The second part determines the path followed by the message until all the destination nodes are reached. Each physical channel is split into a set of virtual channels. If C_1 is the set of original channels and C is the set of all the virtual channels in the network, let C_{1H} be the set of channels belonging to C_1 that connect lower label nodes to higher label nodes and let C_{1L} be the set of channel belonging to C_1 that connect higher label nodes to lower label nodes. Let C_{xyH} be the set of output channels from node x such that each channel belongs to a minimal path from x to y and the label of its destination node is not higher than the label of y. Let C_{xyL} be the set of output channels from node x such that each channel belongs to a minimal path from x to y and the label of its destination node is not lower than the label of y. The routing function designed by the author can use any of these new channels belonging to a minimal path between successive destinations, except when a node with a label higher or lower than the label of the destination node is reached.

Unicast-Based Multicast

McKinley *et. al.* [100, 117, 118] presented efficient algorithms to implement multicast communication in wormhole-routed direct networks, by exploiting the prop-

erties of the switching technology. Minimum-time multicast algorithms have been presented for one-port n-dimensional meshes, tori and hypercubes that use deterministic, dimension-ordered routing of unicast messages. The presented algorithms can deliver a multicast message to $m-1$ destinations in $\lceil log_2 m \rceil$ message passing steps (which is the lower bound on the number of steps needed to broadcast on a one-port architecture), while avoiding contention among the constituent unicast messages. These algorithms are all based on recursive doubling. For each topology there is a corresponding algorithm, and hence the algorithms are called ***U-cube***, ***U-mesh***, and ***U-torus***.

The key to avoiding contention among the constituent messages is the ordering of the destinations. The multicast algorithms take advantage of several simple results regarding message contention and lexicographical ordering, by dimension, of arbitrary sets of nodes. In a 2D mesh, such an ordering is simply a column-wise or row-wise listing of the node addresses. Such an ordering is also called ***dimension order***, denoted by the symbol $<_d$. A sequence of node addresses that are ordered with respect to dimension order is referred to as a ***dimension-ordered chain***. The routes so generated are guaranteed to be arc-disjoint (i.e., they do not share any channel) and thus contention-free.

For the hypercube network in which e-cube routing is used, the symmetry of the topology effectively allows the source node to play the role of the first node in a dimension-ordered chain. The exclusive-or operation, $\oplus$, is used to carry out this task. A sequence $d_1, d_2, \ldots, d_{m-1}$ of hypercube addresses is called a d_0-***relative dimension-ordered chain*** if and only if $d_0 \oplus d_1$, $d_0 \oplus d_2$, ..., $d_0 \oplus d_{m-1}$ is a

dimension-ordered chain. The source can easily sort the $m-1$ destinations into a d_0-relative dimension-ordered chain, $\Phi = d_1, d_2, \ldots, d_{m-1}$. The source may then execute the ***chain algorithm*** using Φ instead of the original addresses. The chain algorithm is a distributed algorithm than can be used to multicast a message from a source node to one or more destinations. The algorithm applies to situations in which the address of the source node is either less than or greater than those of all the destinations according to the $<_d$ relation. The source address and the destination addresses are arranged as a dimension-ordered chain in either increasing or decreasing order. The source node sends first to the destination node halfway across the chain, then to the destination node one quarter of the way across the chain, and so on. Each destination receives a copy of the message from its parent in the multicast tree and may be responsible for forwarding the message to other destinations.

In meshes, symmetry cannot be used for cases in which the source address lies in the middle of a dimension-ordered chain. However, another relatively method has been used to address this problem. In the ***U-mesh*** algorithm, the source and destination addresses are sorted into a dimension-ordered chain, denoted Φ. The source node successively divides Φ in half. If the source is in the lower half, then it sends a copy of the message to the smallest node (with respect to $<_d$) in the upper half. That node will be responsible for delivering the message to the other nodes in the upper half, using the chain algorithm. If the source is in the upper half, then it sends a copy of the message to the largest node in the lower half. The source continues this procedure until Φ contains only its own address.

The ***U-mesh*** algorithm is not contention-free when executed on a torus; however, a simple extension of the dimension-ordered chain has been used to avoid contention in a torus. The source x_s and destinations are sorted into a dimension-ordered chain $\Phi = \{x_0, x_1, \ldots, x_s, \ldots, x_{m-1}\}$. Next, Φ is rotated so that x_s becomes the first element in the chain. The resultant chain $\Phi' = \{x_s, x_{s+1}, \ldots, x_{m-1}, x_0, x_1, \ldots, x_{s-1}\}$ is called an ***R-chain with respect to*** x_s. The ***U-torus*** algorithm takes an R-chain as input and uses the recursive doubling algorithm.

Multi-Packet Multicast

An approach to multicasting with message partitioning, comprising of a family of pipelined multicast algorithms, was proposed by De Coster *et. al.* [23]. The algorithm begins with the source partitioning the message into p packets. The source sends all the packets, in succession, to its first child in the tree. When it has sent the last packet to the first child, it begins sending the packets to the second child. A child stores each packet it receives, and also forwards a copy to its own children. The tree is based on a lexicographical ordering of the source and destinations, but its particular structure can be tailored to the number of nodes m and the number of packets p.

7.2.3 Comparison of Multicast Routing Techniques

Now we present a comparison among the various multicast routing techniques. The comparison is made on the basis of the switching technique used, the topology, the port model, and fault-tolerance. Additional bases such the time complexity of an

algorithm, number of time steps to complete the multicast, and the maximum path length guaranteed by the technique are also used. In these comparisons we follow the organization of Table 7.2.

Sidewalk multicasting [77, 79] guarantees the path length to each destination to be of length at most $H+2F$, where H is the Hamming distance between the source and destination and F is the number of faults. *Lookahead multicasting* [87] can tolerate up to $n-1$ faulty links and the number of additional time steps required in the worst case is $2n$ in an n-dimensional hypercube. The maximum path length is $H + 2\lfloor(\sqrt{8F+1}-1)/2\rfloor$, where F is the number of faults. The computational complexity of the scheme is $O(log_2\ n)$. In *safety level multicasting* [134] the maximum path length in the safety level based scheme is $< H+2$. The computational complexity of the scheme is $O(n)$. The *best-fit* technique [120] has $O(k\ log\ n)$ complexity on an n-dimensional hypercube, where k is the number of destination nodes. *Greedy multicasting* technique [80] has a computational complexity of $O(nk+n^2)$, where k is the number of destinations. The number of time steps required to complete the broadcast is $O(n)$. *Unicast Based* technique [100, 118, 117] can deliver a multicast message to $m-1$ destinations in $\lceil log_2 m \rceil$ message passing steps.

7.3 Summary

In this chapter we have discussed the various routing techniques for broadcast and multicast communication. Different characteristics of each of technique have been given. Also, a brief discussion of the operation of the scheme is described. In many cases, the number of time steps taken by the algorithm for completing the operation,

Table 7.2: Comparison of Multicast Routing Techniques

Technique	Switch. Tech.	Type	Topology	Port Model	Fault Tolerance
Sidewalk Multicast	VCT, SF WH	Mltcst	Hypc	1-port	no. faulty nodes $< n$ Local Info.
Lookahead Multicast	SF WH	Mltcst	Hypc	1-port	no. faulty links $< n$ Limited Global
Safety Level Multicast	SF, VCT WH	Mltcst	Hypc	1-port	no. faulty nodes $< n$ Limited Global
Best-Fit	WH	Mltcst	Hypc	1-port	Non-FT
Greedy Multicast	VCT	Mltcst	Hypc	1-port	non-FT
Base Routing Conforming	WH	Mltcst	k-ary n-cube	$n+1$-port	non-FT
Double Channel	WH	Mltcst	2D-Mesh	n-port	non-FT
Pathlike Mltcst	WH	Mltcst	2D-Mesh	n-port	non-FT
Column Path	WH	Mltcst	2D-Mesh	1-port	non-FT
VC Based Multicast	WH	Mltcst	2D-Mesh	1-port	non-FT
Unicast Based	WH	Mltcst	nD-Mesh Tori, Hypc	1-port	non-FT
Multipacket Multicast	WH	Mltcst	nD-Mesh	1-port	non-FT

the time complexity of the algorithm, or the maximum path length have also been described. Moreover, a comparison has been made between the various broadcast and multicast routing techniques. The comparison has been based on the topology and other features such as the port model, the size of the routing path, etc.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

Providing efficient communication is one of the most important problems in parallel computing and it depends upon the underlying routing scheme. For this reason it is essential to know which routing techniques are suitable and practical. Although an extremely wide number of routing algorithms have been proposed and implemented in hardware and software, it is difficult for a designer of multicomputer networks to choose an appropriate routing algorithm given a particular architectural configuration. For this purpose there is a need for collecting together the various techniques proposed in the literature, classifying them and comparing them. In view of this need, the present work comprises the study of a number of routing schemes along with a proposed model for classifying and comparing them. In addition, various concepts that are fundamental to the area of multicomputer message routing have also been brought together. An attempt is also made to clarify, through examples,

the diverse terminology of this field.

An introduction to multicomputer networks and their various types and characteristics have been presented. The different characteristics of multicomputer networks described here are topology, routing, flow control, and switching. An operational model of multicomputer networks has also been suggested. A discussion on various performance measures used to assess the performance of multicomputer networks has also been conducted. Various properties of the class of adaptive routing techniques have been described in detail. A discussion on the use of routing for broadcast and multicast communication has also been conducted. Different techniques for constructing the broadcast and multicast trees have also been outlined. A description of some of the problems encountered in routing and suggestions to their solution has also been presented.

Various switching techniques used in multicomputer networks have been discussed. Also described in detail is the concept of virtual channels and how they are used in providing deadlock-freedom. We have also compared the switching techniques in the light of their advantages and disadvantages. A study comprising a number of routing techniques for one-to-one routing, fault-tolerant routing, and broadcast and multicast routing has been presented. Different characteristics of each technique have been given. Also, a brief discussion on the operation of the scheme is held. A comparison has been made between the various one-to-one routing techniques, the fault-tolerant routing techniques and the broadcast and multicast routing techniques. The comparison has been based on the proposed classification, the topology and other features such as the port model, complexity, the size of the

routing path, etc.

The study can be helpful to a designer of multicomputer networks in the following manner. After having decided about the topology on which to base the multicomputer, a routing technique needs to be picked for it. At this stage, the designer has to look into the characteristics of the various available techniques. In case one of these existing ones satisfies the design requirements, then it can be implemented. In case none of these schemes is applicable, they may still be helpful in coming up with a new routing strategy for the multicomputer being designed.

8.2 Future Work

As can be seen from the presented classification and literature review that not all the branches of the classification tree have been explored. Specifically, there are only few techniques based on backtracking, especially in k-ary n-cube and mesh networks. What can be suggested as future work is to explore further into those branches of the proposed classification model that have not been studied earlier or there is only scarce study. The comparisons presented here bring forth the unexplored areas of multicomputer routing which may be further pursued. This provides a basis for further investigation and application of these ideas to different topologies, and opens new dimensions for improving and modifying these techniques to obtain newer, hopefully better designs. So, new schemes can be formulated and their performance studied, keeping in mind the drawbacks of previously suggested ones.

Bibliography

[1] George B. Adams III, Dharma P. Agrawal and Howard Jay Siegel, "*A survey and comparison of fault-tolerant multistage interconnection networks*," *IEEE Computer Magazine*, Vol. 20, No. 6, June 1987.

[2] A. Agarwal, "*Limits on interconnection network performance*," *IEEE Transaction on Parallel and Distributed Systems*, Vol. 2, No. 4, October 1991, pp. 398-412.

[3] William C. Athas and Charles L. Seitz, "*Multicomputers : Message passing concurrent computers*," *IEEE Computer Magazine*, Vol. 2, No. 8, August 1988, pp. 9-24.

[4] M. Barnett, D. G. Payne, R. van de Geijn and J. Watts, "*Broadcasting on meshes with wormhole routing*," *Tech. Rep. TR-93-24*, Department of Computer Science, The University of Texas at Austin, November 1993.

[5] M. Barnett, D. G. Payne and R. van de Geijn, "*Optimal broadcasting in mesh-connected architectures*," *Department of Computer Science*, Tech. Rep. TR-91-38, The University of Texas at Austin, December 1991.

[6] Laxmi N. Bhuyan, Qing Yang and Dharma P. Agrawal, "***Performance of Multiprocessor Interconnection Networks***," *IEEE Computer Magazine*, Vol. 22, No. 2, February 1989, pp. 25-37.

[7] Douglas M. Blough and Hongying Wang, "***Cooperative Diagnosis and Routing in Fault-Tolerant Multiprocessor Systems*** ," *Journal of Parallel and Distributed Computing* , Vol. 27, No. 2, June 1995, pp. 205-211.

[8] Rajendra V. Boppana and Suresh Chalsani, "***A comparison of adaptive wormhole routing algorithms***," *Proceedings of the 20th Annual International Symposium on Computer Achitecture*, May 1993, pp. 351-360.

[9] Rajendra V. Boppana and Suresh Chalsani, "***A framework for designing deadlock-free wormhole routing algorithms***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, No. 2, February 1996, pp. 169-183.

[10] Rajendra V. Boppana and Suresh Chalsani, "***Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks***," *IEEE Transactions on Computers*, Vol. 44, No. 7, July 1995, pp. 848-864.

[11] Rajendra V. Boppana and Suresh Chalsani, "***Fault-Tolerant Multicast Communication in Multicomputers***," *International Conference on Parallel Processing*, 1995, Vol. I, pp. 118-125.

[12] Rajendra V. Boppana, Suresh Chalsani and C. S. Raghavendra, "***On Multicast Wormhole Routing in Multicomputer networks***," *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing*, October 1994, pp. 722-729.

[13] Younes M. Boura and Chita Das, "*Fault-Tolerant Routing in Mesh Networks*," *Proceedings of the 1995 ICPP Workshop on Challenges for Parallel Processing*, August 1995, Vol. I, pp. 106-109.

[14] Hasan Çam, "*Worm-through routing: A cautious cut-through switching technique*," *Preprint version*, Department of Computer Engineering, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia.

[15] S. Chalsani and R. V. Boppana, "*Adaptive wormhole routing in tori with faults*," *IEE Proc.-Comput.Digit.Tech.*, Vol. 142, No. 6, November 1995, pp. 386-394.

[16] Ming-Syan Chen and Kang G. Shin, "*Adaptive Fault-Tolerant Routing in Hypercube Multicomputers*," *IEEE Transactions on Computers* , Vol. 39, No. 12, December 1990, pp. 1406-1416.

[17] Ming-Syan Chen and Kang G. Shin, "*Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers*," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 1, No. 2, April 1990, pp. 152-159.

[18] Ming-Syan Chen, "*Distributed routing and task allocation in multicomputer systems*," *Ph.D. Dissertation*, University of Michigan, 1988.

[19] Andrew A. Chien, "*A cost and speed model for k-ary n-cube wormhole routers*," *Proceedings of Hot Interconnects Workshop*, August 1993.

[20] Andrew A. Chien and Jae H. Kim, "***Planar-Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors***," *Journal of the Association for Computing Machinery*, Vol. 42, No. 1, January 1995, pp. 91-123.

[21] Ge-Ming Chiu and Shui-Pao Wu, "***A fault-tolerant routing strategy in hypercube multicomputers***," *IEEE Transactions on Computers*, Vol. 45, No. 2, February 1996, pp. 143-155.

[22] G.-M. Chiu, S. Chalsani and C. S. Raghavendra, "***Flexible, routing criteria for circuit-switched hypercubes***," *Journal of Parallel and Distributed Computing*, Vol. 22, No. 2, August 1994, pp. 279-294.

[23] Luc De Coster, Natalie Dewulf and Ching-Tien Ho, "***Efficient multi-packet multicast algorithms on meshes with wormhole and dimension-ordered routing***," *Proceedings of the 1995 ICPP Workshop on Challenges for Parallel Processing*, August 1995, Vol. III, pp. 137-140.

[24] Robert Cypher and Luis Gravano, "***Requirements for deadlock-free, adaptive packet routing***," *SIAM Journal on Computing*, Vol. 23, No. 6, December 1994, pp. 1266-1274.

[25] Robert Cypher and Luis Gravano, "***Storage efficient, deadlock-free packet routing algorithms for torus network***," *IEEE Transactions on Computers*, Vol. 43, No. 12, December 1994, pp. 1376-1385.

[26] William Dally, "***Network and processor architecture for message-driven computers***," Chapter 3, ***VLSI and Parallel Computation***, Ed. Robert Suaya & Graham Birtwistle, Morgan Kaufmann, 1990, pp. 140-222.

[27] William J. Dally, "***Performance analysis of k-ary n-cube interconnection networks***," *IEEE Transactions on Computers*, Vol. 39, No. 6, June 1990, pp. 775-785.

[28] William J. Dally, "***Virtual-Channel Flow Control***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 3, No. 2, March 1992, pp. 194-205.

[29] William J. Dally and Hiromichi Aoki, "***Deadlock-free adaptive routing in multicomputer networks using virtual channels***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 4, April 1993, pp. 466-475.

[30] William J. Dally and Charles L. Seitz, "***Deadlock-Free Message Routing in Multiprocessor Interconnection Networks***," *IEEE Transactions on Computers*, Vol. C-36, No. 5, May 1987, pp. 547-553.

[31] William J. Dally and Charles L. Seitz, "***The torus routing chip***," *Journal of Distributed Computing*, Vol. 1, No. 3, October 1986, pp. 187-196.

[32] Sivarama P. Dandamudi, ***Hierarchical Hypercube Multicomputer Interconnection Networks***, *Ellis Horwood Limited*, 1991.

[33] Binh Vien Dao, J. Duato and Sudhakar Yalamanchili, "***Configurable flow control mechanisms for fault-tolerant routing***," *Proceedings of 22nd Annual International Symposium on Computer Architecture*, June 1995, pp. 220-229.

[34] Khaled Day and Anand Tripathi, "*A comparative study of topological properties of hypercubes and star graphs*," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 1, January 1994, pp. 31-38.

[35] Jeffrey T. Draper and Joydeep Ghosh, "*A comprehensive analytical model for wormhole routing in multicomputer systems*," *Journal of Parallel and Distributed Computing*,Vol. 23, No. 2, November 1994, pp202-214.

[36] Jeffrey T. Draper and Joydeep Ghosh, "*Multipath E-cube Algorithms (MECA) for Adaptive Wormhole Routing and Broadcasting in k-ary n-cubes*," *Proceedings of the 6th International Parallel Processing Symposium*, March 1992, pp. 407-410.

[37] J. Duato, "*A theory of fault-tolerant routing in wormhole networks*," *Proceedings of the International Conference on Parallel and Distributed Systems*, December 1994.

[38] J. Duato, "*Deadlock-free adaptive routing algorithms for multicomputers: evaluation of a new algorithm*," *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing*, 1991, pp. 840-847.

[39] J. Duato, "*Improving the efficiency of virtual channels with time-dependent selection functions*," *Proceedings of the 4th Parallel Architectures and Languages Europe (PARLE) Conference*, June 1992, pp. 636-650.

[40] J. Duato, B. V. Dao, P. T. Gaughan and S. Yalamanchili, "*Scouting: fully adaptive, deadlock-free routing in faulty pipelined networks*," *Pro-*

ceedings of International Conference on Parallel and Distributed Systems, December 1994, pp. 608-613.

[41] J. Duato, "***A theory of deadlock-free adaptive multicast routing in wormhole networks***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 9, September 1995, pp. 976-987.

[42] S. A. Felperin, L. Gravano, G. D. Pifarré and J. L. C. Sanz, "***Routing techniques for massively parallel communication***," *Proceedings of IEEE* (Special issue on Massively Parallel Computers), Vol. 79, No. 4, April 1991, pp. 488-503.

[43] M. L. Fulgham, R. Cypher and J. L. C. Sanz, "***A comparison of SIMD hypercube routing strategies***," *Proceedings of International Conference on Parallel Processing*, 1991, Vol. III, pp. 236-243.

[44] L. Gargano, A. A. Rescigno and Vaccaro, U., "***Fault-Tolerant Hypercube Broadcasting via Information Dispersal***," *Networks* (Special Issue: Interconnection Networks and Algorithms), Vol. 23, No. 4, July 1993, pp. 271-282.

[45] Patrick T. Gaughan, "***Design and analysis of fault-tolerant pipelined multicomputer networks***," *PhD Thesis*, Computer Systems Research Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, Georgia, USA, May 1994.

[46] Patrick T. Gaughan, and Sudhakar Yalamanchili, "***Adaptive Routing Protocols for Hypercube Interconnection Networks***," *IEEE Computer Magazine*, Vol. 26, No. 5, May 1993, pp. 12-23.

[47] Patrick T. Gaughan, and Sudhakar Yalamanchili, "***A performance model of pipelined k-ary n-cubes***," *IEEE Transactions on Computers*, Vol. 44, No. 8, August 1995, pp. 1059-1063.

[48] Patrick T. Gaughan, and Sudhakar Yalamanchili, "***A family of fault-tolerant routing protocols for direct multiprocessor networks***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 6, May 1995, pp. 482-497.

[49] Chistopher J. Glass and Lionel M. Ni, "***Adaptive routing in mesh-connected networks***," *Proceedings of the 12th International Conference on Distributed Computing Systems*, June 1992, pp. 12-19.

[50] Chistopher J. Glass and Lionel M. Ni, "***Fault-tolerant wormhole routing in meshes***," *Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing*, June 1993, pp. 240-249.

[51] Chistopher J. Glass and Lionel M. Ni, "***Maximally fully adaptive routing in 2D meshes***," *International Conference on Parallel Processing*, August 1992, Vol. 1, pp. 101-104.

[52] Chistopher J. Glass and Lionel M. Ni, "***The turn model for adaptive routing***," *Journal of the Association for Computing Machinery*, Vol. 41, No. 5, September 1994, pp. 874-902.

[53] J. M. Gordon and Q. F. Stout, "***Hypercube message routing in the presence of faults***," *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, January 1988, Vol. I, pp. 318-327.

[54] Albert G. Greenberg and Bruce Hajek, "***Deflection Routing in Hypercube Networks*** ," *IEEE Transactions on Communications* , Vol. 40, No. 6, June 1992, pp. 1070-1081.

[55] D. Grunwald and D. Reed, "***Analysis of Backtracking Routing in Binary Hypercube Computers***," *Tech. Rep. UIUCDCS-R-89-1486*, Department of Computer Science, University of Illinois, Urbana-Champaign, February, 1989.

[56] Qian-Ping Gu and Shietung Peng, "***Finding a Routing Path of Optimal Length in Hypercues with Fault Clusters*** ," *Proceedings of the 7th IASTED/ ISMM International Conference on Parallel and Distributed Computing and Systems*, October 1995, pp. 229-233.

[57] J. Guan, W. K. Tsai and B. Blough, "***An analytical model for wormhole routing in multicomputer interconnection networks***," *Proceedings of the 7th International Parallel Processing Symposium*, April 1993, Vol. II, pp. 1-4.

[58] Frank Thomas Hady, "***A performance study of wormhole-routed networks through analytical modeling and experimentation***," *Ph.D. Thesis*, University of Maryland, College Park, 1993.

[59] Ching-Tien Ho and Ming-Yang Kao, "***Optimal Broadcast in All-Port Wormhole Routed Hypercubes***," IEEE Transactions on Parallel and Distributed Systems, Vol. 6, No. 2, February 1995, pp. 200-204.

[60] Kai Hwang and Fayé Briggs, ***Computer Architecture and Parallel Processinbg***, *McGraw-Hill, Inc.*, 1984.

[61] C. R. Jesshope, P. R. Miller and J. T. Yantchev, "***High performance communications in processor networks***," *Proceedings of the 16th ACM Annual International Symposium on Computer Architecture* , 1989, pp. 150-157.

[62] S. Johnsson and Ching-Tien Ho, "***Optimum Broadcasting and Personalized Communication in Hypercubes***," *IEEE Transactions on Computers*, Vol. 38, No. 9, September 1989, pp. 1249-1268.

[63] Dilip D. Kandlur and Kang G. Shin, "***Traffic routing for multicomputer networks with virtual cut-through capability***," *IEEE Transactions on Computers*, Vol. 41, No. 10, October 1992, pp. 1257-1270.

[64] P. Kermani and L. Kleinrock, "***Virtual cut-through: A new computer communication switching technique***," *Computer Network* , Vol. 3, No. 4, September 1979, pp. 267-286.

[65] Jong Kim and Chita R. Das, "***Hypercube communication delay with wormhole routing***," *IEEE Transactions on Computers*, Vol. 43, No. 7, July 1994, pp.806-814.

[66] Jong Kim and Chita R. Das, "***Modeling wormhole routing in hypercubes***," *Proceedings of International Conference on Distributed Systems*, 1991, pp. 386-393.

[67] Chong-kwon Kim and Daniel A. Reed, "***Adaptive Packet Routing in a Hypercube*** ," *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, January 1988, pp. 625-630.

[68] Jong Kim and Kang G. Shin, "***Deadlock-free fault-tolerant routing in injured hypercubes***," *IEEE Transactions on Computers*, Vol. 42, No. 9, September 1993, pp.1078-1088.

[69] Ji-Yun Kim, Hyunsoo Yoon, Seung Ryoul Maeng and Jung Wan Cho, "***Drop-and-Reroute: A new flow control policy for adaptive wormhole routing***," *International Conference on Parallel Processing*, 1995, vol. I, pp. 60-67.

[70] Jae H. Kim, Ziqiang Liu and Andrew A. Chien, "***Compressionless Routing: A Framework for Adaptive and Fault-Tolerant Routing***," *Proceedings of the 21st IEEE Annual International Symposium on Computer Architecture*, 1994, pp. 289-300.

[71] Smaragda Konstantinidou, "***Adaptive, minimal routing in hypercubes***," 6th MIT Conference on Advanced Research in VLSI, 1990, pp. 139-153.

[72] Smaragda Konstantinidou and Lawrence Snyder, "***The chaos router***," *IEEE Transactions on Computers*, Vol. 43, No. 12, December 1994, pp. 1386-1397.

[73] Smaragda Konstantinidou and Lawrence Snyder, "***The chaos router: Architecture and performance***," *Proceedings of International Symposium on Computer Architecture*, 1991, pp. 212-221.

[74] Smaragda Konstantinidou, "***Priorities in nonminimal, adaptive routing***," *Proceedings of International Conference on Parallel Processing*, August 1992, Vol. 1, pp. 67-71.

[75] Smaragda Konstantinidou and Lawrence Snyder, "*The chaos router: A practical application of randomization in network routing*," *Proceedings of 2nd ACM Symposium on Parallel Algorithms and Architectures*, 1990, pp.21-30.

[76] Vipin Kumar, Ananth Grama, Anshul Gupta and George Karypis, ***Introduction to Parallel Computing***, *The Benjamin/Cummings Publishing Company, Inc.*, 1994.

[77] Youran Lan, "*Adaptive fault-tolerant multicast in hypercube multicomputers* ," *Journal of Parallel and Distributed Computing*, Vo. 23, No. 1, October 1994, pp. 80-93.

[78] Youran Lan, "*A Fault-Tolerant Routing Algorithm in Hypercubes* ," *International Conference on Parallel Processing*,1994, Vol. III, pp. 163-175.

[79] Youran Lan, "*Multicast in faulty hypercubes*," *International Conference on Parallel Processing*,1992, Vol. I, pp. 58-61.

[80] Youran Lan, "*Multicast in faulty hypercubes* ," *Journal of Parallel and Distributed Computing*, Vol. 8, No. 1, January 1990, pp. 30-41.

[81] Youran Lan, "*Interprocessor communication in distributed memory multicprocessors*," *Ph.D. Thesis*, Michigan State University, 1988.

[82] C.R. Lang Jr., "*The extension of object-oriented languages to a homogeneous, concurrent architectures*," *Tech. Rep. 5014*, Dept. of Computer Science, California Institute of Technology, May 1982.

[83] Hyun Suk Lee, Ho Won Kim, Jong Kim and Snuggu Lee, "***Adaptive virtual cut-through as an alternative to wormhole routing***," *International Conference on Parallel Processing*, 1995, Vol. I, pp. 68-75.

[84] Tze Chiang Lee and John P. Hayes, "***A Fault-Tolerant Communication Scheme for Hypercube Computers***," *IEEE Transactions on Computers*, Vol. 41, No. 10, October 1992, pp. 1242-1256.

[85] F. Thomas Leighton, ***Introduction to Parallel Algorithms and Architecture: Arrays, Trees, Hypercubes***, *Morgan Kaufmann Publishers, Inc.*, San Mateo, California, 1992.

[86] Qiang Li, "***Minimum deadlock-free message routing restrictions in binary hypercubes***," *Journal of Parallel and Distributed Computing*, Vol. 15, No. 2, June 1992, pp. 153-159.

[87] Albert C. Liang, Sourav Bhattacharya and Wei-Tek Tsai, "***Fault-Tolerant Multicasting on Hypercubes***," *Journal of Parallel and Distributed Computing*, Vol. 23, No. 3, December 1994, pp. 418-428.

[88] Wei-Kuo Liao and Chung-Ta King, "***Valved routing: Efficient flow control for adaptive nonminimal routing in interconnection networks***," *IEEE Transactions on Computers*, Vol. 44, No. 10, October 1995, pp. 1181-1193.

[89] Chiu-Chuan Lin and Ferng-Ching Lin, "***Minimal fully adaptive wormhole routing on hypercubes***," *Information Processing Letters*, Vol. 50, No. 6, June 1994, pp. 297-301.

[90] Xiaola Lin, Philip K. Mckinley and Lionel M. Ni, "***Deadlock-Free Multicast Wormhole Routing in 2D Mesh Multicomputers***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 8, August 1994, pp. 793-804.

[91] Xiaola Lin and Lionel M. Ni, "***Multicast Communication in Multicomputer Networks***," *Proceedings of the 1990 International Conference on Parallel Processing*, August 1990, Vol. III, pp. 114-118.

[92] Xiaola Lin, Philip K. Mckinley and Lionel M. Ni, "***The message flow model for rotuing in wormhole-routed networks***," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, No. 7, July 1995, pp. 755-760.

[93] Xiaola Lin, Philip K. Mckinley and Lionel M. Ni, "***The message flow model for rotuing in wormhole-routed networks***," *International Conference on Parallel Processing*, 1993, Vol. I, pp. 294-297.

[94] D. H. Linder and J. C. Harden, "***An adaptive and fault-tolerant wormhole routing strategy for k-ary n-cubes***," *IEEE Transactions on Computers*, Vol. 40, No. 1, January 1991, pp. 2-12.

[95] Yen-Wen Lu, Kallol Bagchi, James B. Burr and Allen M. Peterson, "***A comparison of different wormhole routing schemes***," *MASCOT 1994*, 2nd International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems, NC, USA, 1994, pp. 323-328.

[96] Ausif Mahmood, Donald J. Lynch and Roger B. Shaffer, "***Optimal Circuit-Switched Routing in Hypercubes***," *International Conference on Parallel Processing*, 1995, Vol. III, pp. 123-131.

[97] Philip K. McKinley, Yih-jia Tsai and David F. Robinson, "*A Survey of Collective Communication in Wormhole-Routed Massively Parallel Computers*," *Tech. Rep. MSU-CPS-94-35*, Department of Computer Science, Michigan State University, East Lansing, MI, June 1994.

[98] Philip K. McKinley, Yih-jia Tsai and David F. Robinson, "*Collective communication in wormhole-routed massively parallel computers*," *IEEE Computer Magazine*, Vol. 28, No. 12, December 1995, pp. 39-50.

[99] Philip K. McKinley and C. Trefftz, "*Efficient broadcast in all-port wormhole-routed hypercubes*," *Proceedings of the 1993 International Conference on Parallel Processing*, Vol. II, pp. 288-291, 1993.

[100] Philip K. McKinley, Hong Xu, Abdol-Hossein Esfahanian and Lionel M. Ni, "*Unicast-Based Multicast Communication in Wormhole-Routed Networks*," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 12, December 1994, pp. 1252-1265.

[101] P. R. Miller and J. T. Yantchev, "*Developing powerful communication mechanism for distributed memory computer computers from simple and efficient message-routing*," *Proceedings of the 5th IEEE Distributed Memory Computing Conference (DMCC5)*, April 1990, Vol. II, pp. 809-823.

[102] Ilan Newman and Assaf Schuster, "*Hot Potato worm routing via store-and-forward packet routing*," *Journal of Parallel and Distributed Computing*, Vol. 30, No. 1, October 1995, pp. 76-84.

[103] J. Ngai and C. Seitz, "*Adaptive routing in multicomputers*," *Opportunities and constraints of parallel computing*, J. Sanz, Ed., New York: Springer Verlag, 1989.

[104] J. Ngai and C. Seitz, "*A framework for adaptive routing*," Computer Science Department, California Institute of Technology, Tech. Rep. 5246:TR:87, 1987.

[105] J. Ngai and C. Seitz, "*A framework for adaptive routing in multicomputer networks*," *Proceedings of the 1st ACM Symposium on Parallel Algorithms and Architecture*, 1989, pp. 1-9.

[106] John Yee-Keung Ngai, "*A framework for adaptive routing in multicomputer networks*," *PhD Thesis*, Computer Science Department, California Institute of Technology, Tech., May 1989.

[107] Lionel M. Ni and Philip K. McKinley, "*A Survey of Wormhole Routing Techniques in Direct Networks*," *IEEE Computer Magazine*, Vol. 26, No. 2, February 1993, pp. 62-76.

[108] Dhabaleswar K. Panda, "*Issues in Designing Efficient and Practical Algorithms for Collective Communication on Wormhole-Routed Systems*," *Proceedings of the 1995 ICPP Workshop on Challenges for Parallel Processing*, August 1995, Vol. I, pp. 8-15.

[109] Dhabaleswar K. Panda, S. Singhal and Prabhakaran "*Multidestination message passing mechanism conforming to base wormhole routing*

scheme," *Proceedings of Parallel Routing nad Communication Workshop*, May 1994.

[110] Seungjin Park, "***Adaptive Deadlock-Free Wormhole Routing in Faulty Hypercubes*** ," *Proceedings of the 7th IASTED/ISMM International Conference on Parallel and Distributed Computing and Systems*, October 1995, pp. 129-132.

[111] Gustavo D.Pifarré, Luis Gravano, G. Denicolay and Jorge L.C. Sanz, "***Adaptive Deadlock- and Livelock-Free Routing in Hypercube Network*** ," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 11, November 1994, pp. 1121-1139.

[112] Gustavo D. Pifarré, Luis Gravano, Sergio A. Felperin and Jorge L.C. Sanz, "***Fully Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations*** ," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, No. 3, March 1994, pp. 247-263.

[113] R. Ramanathan, and Kang G. Shin, "***Reliable broadcast in hypercube multicomputers*** ," *IEEE Transactions on Computers*, Vol. 37, No. 12, December 1988, pp. 1654-1657.

[114] A. L. Narasimha Reddy and Rich Freitas, "***Fault tolerance of adaptive routing algorithms in multicomputers***," *Proceedings of the 4th IEEE Symposium on Parallel and Distributed Processing*, 1992, pp. 156-161.

[115] Daniel A. Reed and Richard M. Fujimoto, ***Multicomputer networks: Message-based parallel processing***, *The MIT Press*, 1987.

[116] Daniel A. Reed and Dirk C. Grunwald, "***The performance of multicomputer interconnection networks***," *IEEE Computer Magazine*, Vol. 20, No. 6, June 1987, pp. 63-73.

[117] David F. Robinson, Dan Judd, Philip K. McKinley, and Betty H. C. Cheng, "***Efficient multicast in all-port wormhole-routed hypercubes***," *Journal of Parallel and Distributed Computing*, Vol. 31, No. 2, December 1995, pp. 126-140.

[118] David F. Robinson, Philip K. McKinley, and Betty H. C. Cheng, "***Optimal multicast communication in wormhole-routed torus networks***", *Proceedings of the 1994 International Conference on Parallel Processing*, August 1994, Vol. I, pp. 134-141.

[119] Y. Saad and M. Schultz, "***Topological Properties of Hypercubes***," *IEEE Transactions on Computers*, Vol. 37, No. 7, July 1988, pp. 867-872.

[120] Thomas J. Sager and Bruce M. McMillin, "***A fast $O(k)$ multicast message routing algorithm***," *Department of Computer Science*, University of Missouri, Rolla, USA, Tech. Rep. CSC-90-2, March 1990.

[121] Loren Schwiebert and D. N. Jayasimha, "***Optimal Fully Adaptive Minimal Wormhole Routing for Meshes***," *Journal of Parallel and Distributed Computing*, Vol. 27, No. 1, May 1995, pp. 56-70.

[122] A. Sengupta and S. Bandyopadhyay, "***Deadlock-Free Routing in k-ary Hypercube Network in Presence of Processor Failures*** ," *Information Processing Letters*, Vol. 34, No. 6, May 1990, pp. 323-328.

[123] Charles Seitz, "***Concurrent architectures***," Chapter 1, ***VLSI and Parallel Computation***, Ed. Robert Suaya & Graham Birtwistle, Morgan Kaufmann, 1990, pp. 1-84.

[124] George D. Stamoulis, "***Routing and Performance Evaluation in Interconnection Networks***," *Ph.D. Thesis*, Massachusetts Institute of Technology, June 1991.

[125] Harold Stone, ***High-Performance Computer Architecture***, *Addison-Wesley Publishing Co.*, 1987.

[126] Chien-Chun Su and Kang G. Shin, "***Adaptive deadlock-free routing in multicomputers using only one extra virtual channel***," *International Conference on Parallel Processing*, 1993, Vol. 1, pp. 227-231.

[127] Robert Suaya and Graham Birtwistle, ***VLSI and Parallel Computation***, Morgan Kaufmann, 1990.

[128] Young-Joo Suh, Binh Vien Dao, José Duato and Sudhakar Yalamanchili, "***Software based fault-tolerant oblivious routing in pipelined networks***," *International Conference on Parallel Processing*, 1995, Vol. 1, pp. 101-105.

[129] H. Sullivan and T. R. Bashkow, "*A large scale, homogeneous, fully distributed parallel machine*," *Proceedings of the 4th Annual Symposium on Computer Architecture*, March 1977, Vol. 5, pp. 105-124.

[130] Andrew S. Tanenbaum, ***Computer Networks***, 2nd Edition, Prentice-Hall, 1989.

[131] L. Valiant, "*A scheme for fast parallel communications*," *SIAM Journal on Computing*, Vol. 11, No. 2, May 1982, pp. 350-361.

[132] L. Valiant, "*General purpose parallel architectures*," ***Handbook of Theoretical Computer Science***, J. van Leeuwen, Ed., Amsterdam, The Netherlands: North-Holland, 1988.

[133] L. Valiant and G. Brebner, "*Universal schemes for parallel communication*," *ACM STOC*, 1981, pp. 263-277.

[134] Jie Wu, "*A limited-global-information based multicasting scheme for faulty hypercubes*," *IEEE Transactions on Computers*, Vol. 44, No. 9, September 1995, pp.1162-1167.

[135] Jie Wu, "*Unicasting in Faulty Hypercubes Using Safety Levels*," *International Conference on Parallel Processing*, 1995, Vol. III, pp. 132-136.

[136] J. T. Yantchev and C. R. Jesshope, "*Adaptive, low latency, deadlock-free packet routing for network of processors*," *IEE Proceedings*, Vol. 136, Pt. E, No. 3, May 1989, pp. 178-186.

[137] S. D. Young and S. Yalamanchili, "***Adaptive routing in generalized hypercube architectures***," *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing*, 1991, pp. 564-571.

Vita

- Farooq Ashraf
- Born on March 26, 1972, at Karachi, Pakistan.
- Received Bachelor of Science degree in Computer Science from School of Engineering at The City College of The City University of New York, New York, USA, 1994.
- Joined the Information and Computer Sciences Department, King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, as a Research Assistant in October, 1994.
- Completed Masters Degree in Computer Science from King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, in June 1996.